Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática

Ano 2017

NELSON MOREIRA CABRAL

TERMINAL DE TV

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor António José Nunes Navarro Rodrigues, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Dedico este trabalho à minha família pelo incansável apoio e motivação.

O júri

Presidente	Professor Doutor Tomás António Mendes Oliveira e Silva professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
Arguente	Professor Doutor João Miguel Duarte Ascenso professor auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores do Instituto Superior Técnico de Lisboa
Orientador	Professor Doutor António José Nunes Navarro Rodrigues professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Agradecimentos

Venho, deste modo, expressar o meu reconhecimento a todos aqueles que, direta ou indiretamente, contribuíram para a concretização deste trabalho:
Em primeiro lugar, ao meu orientador Professor Doutor António José Nunes Navarro Rodrigues pelo acompanhamento que me deu, mostrando sempre disponibilidade para responder a todas as minhas questões e encaminhar o meu trabalho através de preciosas recomendações;
A toda a minha família que, desde sempre, me ensinou a lutar e a trabalhar arduamente para atingir os meus objetivos;
Ao Nuno Coelho e ao David Marques, pelas suas sugestões, reflexões, disponibilidade e amizade, sem as quais este trabalho não teria sido possível;
A todos aqueles que, na impossibilidade de referir os seus nomes, estiveram sempre a meu lado com uma palavra de apoio.

palavras-chave

Terminal, IPTV, Escalabilidade de Vídeo, Visualização, Protocolos de Comunicação, Interactividade, Mobilidade

resumo

Os sistemas modernos de multimédia proporcionam uma experiência muito rica em termos de conteúdos tanto de vídeo como de áudio bem como de aplicações para a web.

O presente trabalho propõe o desenvolvimento de um terminal que suporte essa nova experiência, do tipo Internet TV, em consequência do projeto Europeu FP6 SUIT, com a capacidade de receber sinais de difusão bem como de vídeo a pedido (VoD), VoIP e serviços de internet móvel. O objetivo é ter a capacidade de descodificar até HDTV 1080i/720p/50Hz, demonstrar escalabilidade de vídeo e ter interatividade através de serviços como o VoIP e a Internet.

Avaliar em termos de *hardware* quais seriam os recursos mínimos para que fosse possível descodificar HDTV, bem como a robustez do descodificador em situações críticas como a mobilidade. Por isso, foi realizada uma campanha de medidas de campo de forma a avaliar a robustez dos esquemas de transporte dos conteúdos. Foi também avaliada a capacidade de ultrapassar erros de transmissão e desse modo tornar o terminal bastante robusto.

keywords

Terminal, IPTV, Video scalability, Rendering, Comunications Protocols, Interactivity, Mobility

abstract

Modern multimedia systems provide a rich experience in terms of both video and audio content as well as web applications.
The present work proposes the development of a terminal that supports this new Internet TV experience, as a consequence of the FP6 European SUIT project, with the capacity to receive broadcast signals as well as Video on Demand (VoD), VoIP and mobile internet services. The goal is to have the ability to decode up to 1080i / 720p / 50Hz HDTV, demonstrate video scalability and have interactivity through services like VoIP and the Internet.
Evaluate in terms of hardware what would be the minimum resources for decoding HDTV as well as the robustness of the decoder in critical situations such as mobility. Therefore, a campaign of field measurements was carried out in order to evaluate the robustness of content transport schemes. Also evaluated was the ability to overcome transmission errors and thereby make the terminal quite robust.

Índice

Índice de Figuras

ACRÓNIMOS

AP	Access Point
CIF	Common Intermediate Format
CINR	Carrier to Interference + Noise Ratio
DNS	Domain Name System
DVB	Digital Video Broadcasting
DVBSTP	DVB SD&S Transport Protocol
DVB-RCT	Digital Video. Broadcasting Return Channel – Terrestrial
DVB-T	Digital Video Broadcasting-Terrestrial
EPG	Eletronic Programming Guide
GOP	Group of Pictures
GSM	Global System for Mobile Communications
GPU	Graphics Processing Unit
HDCP	High-bandwidth Digital Content Protection
HDMI	High-Definition Multimedia Interface
HDTV	High Definition Television
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IP	Internet Protocol
JSVM	Joint Scalable Video Model
LMB	Live Media Broadcast
MAC	Media Access Control
Mbps	Megabit per second
MDC	Multiple Description Coding
MHP	Multimedia Home Platform
MPEG	Moving Picture Experts Group
MTU	Maximum Transmission Unit

NAL	Network Abstraction Layer
NALU	Network Abstraction Layer Unit
OpenGL	Open Graphics Library
PDA	Personal Digital Assistant
PIP	Picture in Picture
QCIF	Quarter Common Intermediate Format
QoS	Quality of Service
RCIC	Return Channel Information Collector
RF	Radio Frequency
RSVP	Resource Reservation Protocol
RTP	Real Time Protocol
RTCP	Real Time Control Protocol
RTSP	Real Time Streaming Protocol
SDC	Single Description Coding
SDL	Simple DirectMedia Layer
SDTV	Standard Definition Television
SDP	Session Description Protocol
SDPng	Session Description Protocol, new generation
SD&S	Service Detection & Selection
STB	Set-Top-Box
SVC	Scalable Video Coding
SUHDTV	Super Ultra HDTV
SUIT	Scalable, Ultra-fast and Interoperable Interactive Television
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
UED	Usage Environment Description
USB	Universal Serial Bus
VoD	Video on Demand
VoIP	Voice over Internet Protocol
WiMAX	Worldwide Interoperability for Microwave Access
XML	Extensible Mark-up Language

CAPÍTULO 1 - Introdução e motivação

"Engenharia não é meramente saber e ter conhecimento, como uma enciclopédia; engenharia não é meramente análise; engenharia não é meramente possuir a capacidade de descobrir soluções elegantes para problemas; engenharia é colocar em prática a arte organizada de forçar as mudanças tecnológicas...os engenheiros operam na interface entre a ciência e a sociedade..." [1]

O terminal TV é um trabalho de investigação e desenvolvimento resultante do projeto SUIT- *Scalable Ultra-fast Interactive Television* do 6º programa quadro da união europeia. Foi o primeiro projeto liderado pelo instituto de Telecomunicações-Pólo de Aveiro e decorreu entre 2006 e 2008.

O trabalho desenvolvido dividiu-se em duas fases. Numa primeira fase, maioritariamente de investigação, foram identificados os requisitos do terminal de TV, e a melhor maneira de os implementar tendo em conta as tecnologias que melhor se adequavam ao problema. A segunda fase correspondeu à implementação propriamente dita do terminal de TV, onde todo o sistema foi desenvolvido precisamente para corresponder às especificidades do projeto.

1.1 Enquadramento do trabalho

A televisão é vista, por muitos, como a invenção mais importante do século XX. Perceber a sua utilização assim como os vetores da evolução tecnológica, social e económica é da maior importância para qualquer entidade que desenvolve soluções para entrega de conteúdo televisivo.

Embora, após as evoluções crescentes na indústria de telecomunicações, a televisão ainda continua a ser o meio de comunicação mais eficiente para educar, informar e divertir os

[1] *Dean Gordon Brown* (1907-1996).

telespectadores. Os televisores ainda representam os aparelhos eletrónicos domésticos mais populares no mundo. Com a sua massificação na metade da década de 50, melhoramentos não faltaram à TV: a evolução para TV a cores, a invenção do telecomando, até aos mais recentes progressos rumo à digitalização do sinal resultaram nas mais recente tecnologia de *High Definition Television* (HDTV), o SUHDTV de 4K de resolução, já para não falarmos das funcionalidades incorporadas nas SMART TV.

Uma última novidade nesta área, é o conceito de *Internet Protocol Television* (IPTV) que é, em grande parte, facilitado pelo bem conhecido protocolo *Internet Protocol* (IP) e a evolução das plataformas que permitem a oferta de serviços de transmissão de vídeo utilizando uma infra-estrutura IP convergente. A conjunção do binómio IP+TV é expressão do conceito da convergência multimédia: voz, vídeo e dados. Por vezes equivocado com o serviço de Internet TV em que se utiliza o PC ligado à internet pública, também conhecido como *Over-the-Top*, para ver programas e canais de TV, o serviço IPTV permite o uso de um rede banda larga privada, por exemplo fibra ótica, DVB-RCT, WiMAX, entre outros, para a entrega de conteúdo de TV, pelo operador, com garantia de qualidade de serviço (QoS) e potencialmente acrescido de serviços interativos.

Os serviços IPTV aproveitam-se da capacidade e da natureza bidirecional das redes de banda larga para oferecerem serviços interativos e responder à crescente exigência dos requisitos dos assinantes. Assinantes estes, que procuram cada vez mais serviços individualizados e personalizados que os permitam escolher o quê, e quando querem assistir. Vídeo a pedido também conhecido como *Vídeo on Demand* (VoD), gravação seletiva de conteúdos, aplicações interativas com os jogos *online* e capacidade de poderem assistir ao seu programa ou série em diferido fazem parte da lista de alguns serviços IPTV que ilustram o sentido das preferências dos utilizadores finais.

O IPTV abre novas possibilidades de negócio aos operadores de telecomunicações, como é o caso do *Triple Play* (voz, internet e telefone) ou *Quadruple Play* (*Triple Play* + mobilidade). Nesta área existe a obrigação de desenvolver novos produtos que vão de encontro às pretensões dos utilizadores finais. A TV móvel com qualidade *High Definition* (HD) e 4K será sem dúvida um novo marco, que ficará na história. Além disso ter disponível *internet* e *voice over internet protocol* (VoIP) em qualquer lado, a qualquer momento, será sem dúvida um produto muito atraente para o consumidor e vantajoso para os operadores de telecomunicações.

1.2 Objetivos

O objetivo desta tese é o desenvolvimento de um terminal de televisão (IPTV), no âmbito de um projeto europeu denominado *Scalable, Ultra-fast and Interoperable Interactive Television* (SUIT) [1] do 6º programa quadro da união europeia, que seja capaz de fornecer ao utilizador conteúdos fornecidos pelos operadores de telecomunicações usando a rede

DVB-T/DVB-H/DVB-RCT e WiMAX. Conteúdos tais como TV, *Internet* e VoIP, fixo e móvel e com qualidade de serviço fazendo uso do WiMAX de forma a complementar o DVB em caso de falta de cobertura.

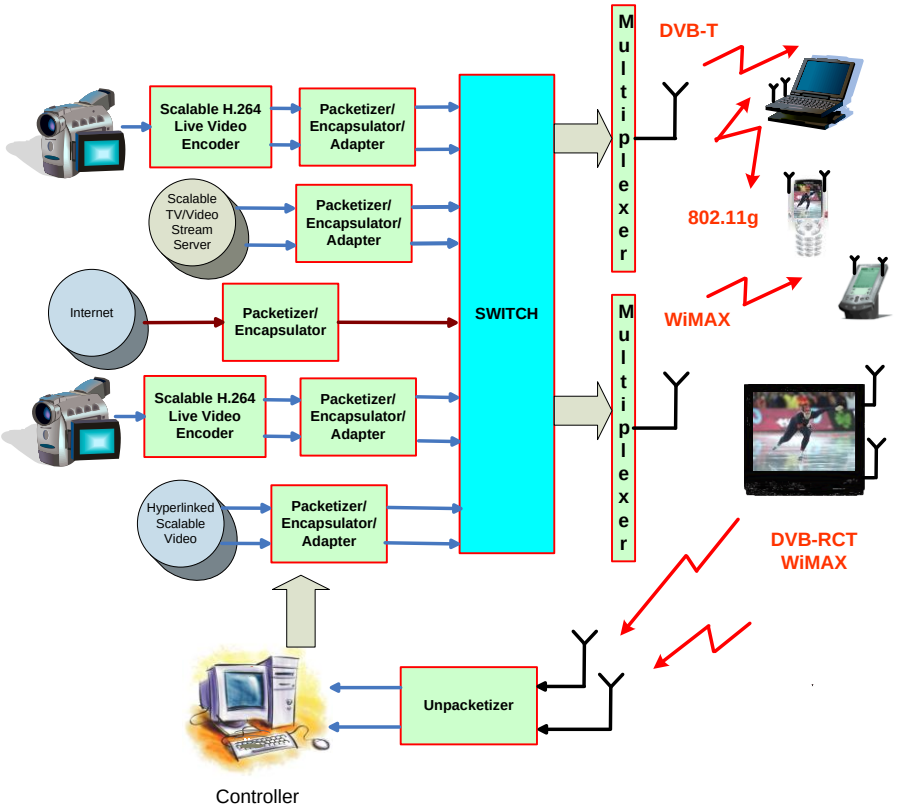


Figura 1 - Arquitetura SUIT.

Como pode ser observado na Figura 1, o projeto SUIT, visa a distribuição de vídeo escalável *Scalable Video Coding* (SVC) [2] usando as redes DVB-T [3] e WiMAX [4] em tempo real e pré-gravado, para que seja possível visualizar o vídeo numa multitude de terminais de diferentes capacidades, tais como terminais móveis, portáteis, telemóveis bem como numa televisão SDTV ou HDTV. Convém referir que existe uma *gateway* no caso em que exista cobertura com WiMAX, responsável pela combinação dos dois sinais difundidos pelo DVB e pelo WiMAX e que posteriormente envia essa combinação para o terminal de TV. Permite também a distribuição de serviços, como o VoD (*unicast*), o VoIP e a *internet* mas usando apenas a rede WiMAX.

A utilização de vídeo escalável permite ao operador minimizar os custos, apenas necessitando de um *playout* evitando um codificador para cada resolução. Para além disso, a alta compressão, recorrendo ao H.264/SVC [5], implica menor largura de banda, ou seja, podendo multiplexar mais canais num mesmo *multiplexer*. Hoje em dia o número de dispositivos que requerem capacidades e resoluções diferentes na reprodução de vídeo está em crescimento, porém, a largura de banda não está a aumentar na mesma proporção. O

codificador de vídeo H.246/SVC possibilita codificar um determinado vídeo em várias resoluções temporais e espaciais e qualidades diferentes através da aplicação da definição de camada, ao passo que a codificação em apenas uma camada fornece apenas um nível de resolução e qualidade. Assim sendo, a codificação em SVC tem como principal objetivo a utilização de uma menor largura de banda comparando com a transmissão de vídeo simultânea. O SVC foi idealizado como uma extensão do H.264/MPEG 4-AVC, e assim a maioria dos componentes do H.264/MPEG 4-AVC são utilizados, de acordo com o especificado na norma.

Finalmente, tratando-se de uma rede IP, pode fornecer serviços interativos VoD, telefone (VoIP) e *Internet*.

A qualidade de serviço (QoS), no que diz respeito à difusão, é apenas fornecida quando está disponível a rede WiMAX, por essa razão devemos distinguir dois cenários:

- Cenário Urbano: Neste cenário haverá a rede WiMAX que permitirá garantir a qualidade de serviço (QoS).
- Cenário Rural: Neste cenário não haverá rede WiMAX, logo não haverá garantia de qualidade de serviço (QoS).

Graças à tecnologia DVB-RCT [6], não havendo cobertura de WiMAX na zona, o retorno pode ser realizado por esta tecnologia.

Assim, definem-se como objetivos:

- Criação de um terminal TV (*set-top box* ou STB) capaz de descodificar vídeo escalável até à resolução HD, usando como ferramenta de desenvolvimento o *visual studio* C/C++ e como *hardware* um mini-PC.
- O terminal TV deve ser capaz de descodificar e visualizar vídeo com escalabilidade, de forma a emular o funcionamento de diversos tipos de equipamentos nomeadamente dispositivos móveis/portáteis.
- Testar o correto funcionamento em ambientes adversos de todos os componentes do terminal.
- Avaliar a vantagem de ter um terminal de TV com diversidade de tecnologia de transmissão.

1.3 Organização da dissertação

Esta dissertação é composta por cinco capítulos e um anexo. No capítulo presente é apresentada a motivação subjacente a esta dissertação sob a forma de uma introdução ao conceito de IPTV sobre redes de alto débito e uma breve descrição do projeto SUIT. No capítulo dois é apresentado o estado da arte de terminais de TV com foco em STBs, tendo em conta, também, a oferta dos operadores de telecomunicações. No capítulo três são apresentados os protocolos, possíveis, utilizados num terminal de TV, descrevendo-os em detalhe. No capítulo quatro é apresentado o trabalho desenvolvido conducente ao protótipo final, o seu funcionamento e descrição das funcionalidades do terminal de TV bem como os resultados obtidos. No capítulo cinco é apresentado um conjunto de breves conclusões relativas ao trabalho efetuado. Os anexos consistem em partes do código fonte desenvolvido em *visual studio* C/C++ entre outros.

CAPÍTULO 2 - Estado da arte de STBs de TV

No capítulo 2 descreve-se o estado da arte relacionado com os terminais de TV, portanto, o trabalho de investigação efetuado assentou numa análise de alguns terminais de TV já existentes, bem como por uma definição de quais as tecnologias disponíveis e sobre as quais o sistema iria alicerçar-se. Facilmente se conclui sobre a inexistência no mercado de STBs com a capacidade de descodificação H.264/SVC.

2.1 Análise de terminais de TV

Neste ponto encontra-se uma análise detalhada efetuada a alguns terminais de TV já existentes. Uma análise preliminar mostrou-nos que não existiam no mercado, naquele momento, terminais de TV que combinassem WiMAX e DVB-T e com DVB-RCT, ou seja, com canal de retorno.

Assim sendo, estudamos as principais características de terminais de TV segundo os seguintes critérios:

1- Fornecerem ou não serviços semelhantes ao do terminal de TV a desenvolver;
2- Sejam baseados no sistema operativo Windows ou LINUX;
3- Funcionem no sistema IPTV.

Os terminais de TV estão, cada vez mais, a converterem-se em autênticos computadores. Muitos dos terminais de TV já vêm equipados com o sistema operativo, *open source*, LINUX, disco rígido, o que permite a gravação de conteúdos, memória RAM significativa e várias saídas para periféricos, as mais comuns USB e o HDMI. O protocolo de comunicação RS-232, que existia nos terminais de TV antigos, deixou de existir nos mais modernos. Os terminais atuais estão também equipados com saída de áudio digital, vídeo composto, entre outras. Contudo, apenas serão descritos alguns exemplos, os mais semelhantes, de terminais TV que estão disponíveis no mercado para o consumidor final.

Figura 2 - Terminal Pandora DVB-T.

Caraterísticas do terminal Pandora DVB-T H.264 :

- ✓ Capacidade de recepção HDTV e SDTV
- ✓ VoD
- ✓ Wi-Fi ou cabo de rede LAN
- ✓ H.264 MPEG-4 AVC
- ✓ Sistema operativo uCLinux

Figura 3 - Terminal Dreambox DM800 PVR DVB-T.

Caraterísticas do terminal Dreambox DM800 PVR DVB-T**:**

- ✓ Processador 300 MHz MIPS
- ✓ LINUX
- ✓ Descodificação por *hardware* H.264
- ✓ 2 x USB 2.0
- ✓ 64 MByte Flash, 256 MByte RAM

Figura 4 - Terminal AB IPBox 900HD DVB-T.

Caraterísticas do terminal AB IPBox 900HD DVB-T**:**

- ✓ LINUX
- ✓ MPEG-4 AVC/H.264
- ✓ HDMI com HDCP
- ✓ USB 2.0
- ✓ Formato de resolução: 1080i, 720p, 576i

Figura 5 - Terminal MVision HD200.

Caraterísticas do terminal MVision HD200 DVB-T**:**

- ✓ Suporta MPEG-2, MPEG-4, H.264 e é compatível com DVB
- ✓ Várias saídas HDMI
- ✓ USB 2.0
- ✓ *User friendly* OSD

Figura 6 - Terminal Motorola VIP1616T HD DVB-T/IP.

Caraterísticas do terminal Motorola VIP1616T HD DVB-T/IP**:**

- ✓ MPEG-4 AVC (H.264),
- ✓ 128MB DRAM
- ✓ HDMI com HDCP
- ✓ VoD
- ✓ *Multicast*
- ✓ SDTV e HDTV

Figura 7 - Terminal DTR-1101-EU/TW Mobile DVB-T diversidade.

Caraterísticas do terminal DTR-1101-EU/TW Mobile DVB-T diversidade**:**

- ✓ DIBCOM DIB3000MC (*front-end* com diversidade)
- ✓ SDTV
- ✓ Descodificação MPEG-II

Figura 8 - Terminal ADSL.

Caraterísticas do terminal ADSL**:**

- ✓ DVR
- ✓ Suporta vários *CoDecs* de vídeo incluindo o MPEG-2, MPEG-4 Part 10/H.264
- ✓ 80 GB disco
- ✓ Compatibilidade com 100BaseT *Ethernet* para vídeo IP e conteúdo de dados
- ✓ Saída de vídeo HDTV, amostragem em múltiplos formatos (incluindo 1080i, 720p, 480i, 480p)
- ✓ HDMI com HDCP
- ✓ VoD

Figura 9 - Terminal FIBRA.

Caraterísticas do terminal FIBRA**:**

- ✓ DVR
- ✓ Suporta vários *CoDecs* de vídeo incluindo o MPEG-2, MPEG-4 Part 10/H.264
- ✓ 500 GB disco
- ✓ Compatibilidade com 100BaseT *Ethernet* para vídeo IP e conteúdo de dados data content
- ✓ Saída de vídeo HDTV, amostragem em múltiplos formatos (Ultra HD 4K, incluindo 1080i, 720p, 480i, 480p)
- ✓ HDMI com HDCP
- ✓ VoD

Figura 10 - Terminal IPTV DVB-H SGH-P960.

Caraterísticas do terminal IPTV DVB-H SGH-P960**:**

- ✓ DVB-H
- ✓ QVGA 240x320
- ✓ TV-Out

Figura 11 - Nexbox A1.

Caraterísticas do terminal Nexbox A1**:**

- ✓ *Android* 6.0
- ✓ SoC octa-core Amlogic S912
- ✓ 2 GB RAM
- ✓ 16 GB Disco
- ✓ 4K VP9 por *hardware*
- ✓ H.264/H.265 1080p@60fps
- ✓ HDMI 2.0, *Ethernet* RJ-45, SPDIF, 2x USB 2.0, AV, cartão SD
- ✓ Wi-Fi e *Bluetooth*

O último terminal baseado no sistema operativo *Android* não existia na altura em que foi desenvolvido o Terminal TV do projeto SUIT. Todavia, estes terminais não incluem sintonizadores de WiMAX e não conseguem descodificar SVC.

2.2 Análise da tecnologia dos operadores de TV

A finalidade desta análise era a de, antes de se iniciar a implementação do terminal de TV, indagar o mercado e considerar soluções que dão melhores garantias de fiabilidade e robustez, e ao mesmo tempo possam ser implementados no terminal de TV de forma a respeitar requisitos de funcionamento já analisados. Da análise feita aos terminais de TV disponíveis no mercado, verifica-se que os operadores apenas dispõem canal de retorno no ADSL, na FIBRA e no GSM para a transmissão de canais de TV sobe IP. A escalabilidade

do vídeo é feita consoante o meio do canal para a transmissão dos canais de TV, ou seja, da largura de banda disponível.

Os operadores de telecomunicações desde muito cedo que oferecem serviços VoD, *internet* e VoIP. Por satélite os canais são transmitidos em MPEG-2 vídeo e mais recentemente em MPEG-4 AVC, de forma a ser possível a transmissão de mais canais e serviços usando a mesma largura de banda que o MPEG-2 vídeo. Porém, com a tecnologia disponível por satélite, os operadores apenas podem oferecer serviço de televisão e VoD, este último sem a possibilidade controlo (*forward*, *reverse*, *pause*, etc…), isto porque, nesta tecnologia não existe canal de retorno. O serviço de *electronic programming guide* (EPG) é fornecido.

O sistema mais dominante é o ADSL, em que os operadores oferecem os serviços triplos (TV, *Internet* e Voz) e mais recentemente os serviços quádruplos (TV, *Internet*, Voz e Móvel). A transmissão dos canais de televisão é feita sob IP, permitindo que o VoD tenha a possibilidade de pausa, fazendo com que o serviço seja um autêntico videoclube. Como no satélite o serviço de EPG também é fornecido.

Recentemente a FIBRA começou a ser massificada, permitindo aos operadores oferecer mais e melhores serviços. Canais em *Full* HD começaram a aparecer em grande número, bem como uma lista maior de filmes e séries para serem vistos por VoD.

A mobilidade surge com a tecnologia GSM, o 3G e ultimamente o 4G. Este permite ver alguns canais de televisão em mobilidade. Existe várias limitações a este serviço no que diz respeito à cobertura 3G e 4G.

Á semelhança da plataforma satélite, o DVB-T não oferece canal de retorno, isto porque os recetores não têm a capacidade de transmissão, possuem apenas capacidade de receção, apesar de no projeto SUIT tivéssemos usado DVB-RCT. Com o 1º dividendo e em breve com o 2º dividendo, a probabilidade de utilização de um canal de 8MHz de largura de banda para DVB-RCT diminui consideravelmente. O 4G é presentemente uma boa alternativa para o canal de retorno para o sistema DVB-T/T2.

CAPÍTULO 3 - Protocolos de suporte

3.1 Descrição de serviços e da sessão

No capítulo anterior descrevemos o estado da arte em termos de terminais de TV baseados em STB e as tecnologias utilizadas pelos operadores de telecomunicações. Neste capítulo, vamos descrever os protocolos de rede e de sessão implementados no terminal TV do projeto SUIT.

3.1.1 SDP

Ao iniciarem-se conferências multimédia, por exemplo, chamadas VoIP, *streaming* de vídeo e/ou áudio, ou outro tipo de sessões, existe a necessidade de transmitir os detalhes da sessão multimédia, os endereços de transporte, e outras descrições da sessão de metadados para os participantes nessa sessão, do *playout* para o terminal de TV.

O protocolo de descrição de sessão (SDP) [7] prevê uma representação padrão para esse tipo de informação, independentemente da forma como essa informação é transportada. O SDP é meramente um formato para a descrição da sessão não incorpora, conforme referido, um protocolo de transporte, e destina-se a usar diferentes protocolos de transporte, conforme o mais adequado, incluindo o protocolo de anúncio de sessão (SAP) [8], Protocolo de Inicialização (*Session Initiation*) [9], RTSP [10], qualquer tipo de MIME (*Multipurpose Internet Mail Extension*) que pode ser descrita de forma idêntica à facilidade do *e-mail* suportar todos os tipos de anexos em mensagens e o Protocolo de Transporte Hipertexto.

O objetivo do SDP é a de transmitir informações sobre o fluxo de média em sessões multimédia, a fim de permitir que os beneficiários de uma sessão de multimédia possam participar na sessão. Uma sessão multimédia, para este efeito, é definida como um conjunto de fluxos de média que existem por um período de tempo.

Assim as informações SDP incluem:

• Nome da sessão e a finalidade
• Tempo(s) em que a sessão está ativa
• Informação de como receber os fluxos de média (endereços, portas, formatos, etc)

Como os recursos necessários para participar numa sessão podem ser limitados, algumas informações adicionais poderão ser desejáveis:

• Informação sobre a largura de banda a ser utilizado pela sessão
• Informação de contacto do responsável pela sessão

Em geral, o SDP deve transmitir informações suficientes para permitir que um terminal de TV participe numa sessão.

O formato de uma sessão SDP consiste de uma série de linhas e de texto na seguinte forma:

<*código*> = <*valor*>

onde <*código*> deve ser um caracter *case-sensitive*, ou seja, sensível às letras maiúsculas e minúsculas e <*valor*> é um texto estruturado em que o formato depende do <*código*>. Em geral, <*valor*> é um número de campos delimitados por um único caracter, o espaço, ou uma frase de formato livre, cujos caracteres também são *case-sensitive*, ou seja, sensível às letras maiúsculas e minúsculas a menos que exista um campo específico que define o contrário.

Um SDP é inteiramente textual utilizando a norma ISO 10646 que especifica um conjunto de caracteres que são codificados em UTF-8. No SDP o nome do campo e atributo devem usar apenas o US-ASCII subconjunto do UTF-8, mas os campos textuais que atribuem valores podem utilizar todo o conjunto de caracteres ISO 10646. Os campos e o valor dos atributos que usam a totalidade do set de caracteres UTF-8 nunca são diretamente comparados, por isso não existe a necessidade da normalização para UTF-8. A forma textual, em oposição com a codificação binaria como é ASN.1 ou XDR, foi escolhida para melhorar a portabilidade, permitir uma maior variedade de protocolos de transporte e uma maior flexibilidade nas ferramentas existentes que permitam gerar e processar sessões de multimédia.

Algumas linhas em cada descrição de sessão são necessárias e algumas são opcionais, mas todas devem aparecer exatamente na ordem aqui indicada (a forma fixa aumenta

grandemente a deteção de erro permitindo a utilização de um simples *parser*). Os códigos opcionais estão marcados com um "*".

Descrição da sessão
- v = (versão do protocolo)
- o = (proprietário / criador e identificador sessão).
- s = (nome da sessão)
- i =* (informação da sessão)
- u =* (URL da descrição)
- e =* (endereço de e-mail)
- p =* (número de telefone)
- c =* (informação da ligação)
- b =* (informação da largura de banda)
- z =* (adaptações no fuso horário)
- k =* (chave de encriptação)
- a =* (atributos de sessão (0 ou mais linhas))

Descrição do tempo
- t = (intervalo de tempo em que a sessão está ativa)
- r =* (número de vezes a repetir (0 ou mais linhas))

Em particular, a descrição da sessão de multimédia inclui:

• O tipo de média (áudio, vídeo, etc)
• O protocolo transporte (RTP / UDP / IP, H.320, etc)
• O formato da média (vídeo H.261, H.264 vídeo, vídeo MPEG, etc)

Para uma sessão *multicast*, as seguintes informações também são enviadas:

• Endereço *multicast*
• Porta de transporte

Este endereço e a porta, são o endereço e porta de destino do fluxo *multicast*, se for enviado, recebido, ou ambos os casos.

Para uma sessão *unicast*, as seguintes informações também são enviadas:

• Endereço remoto para envio do fluxo de dados de multimédia
• Porta de transporte para o endereço de contacto

A semântica deste endereço e porta dependem da média e do protocolo de transporte definido. Por defeito, este é o endereço e a porta remota para a qual os dados são enviados, e o endereço e a porta local no qual são recebidos os dados. No entanto, alguns serviços de multimédia podem definir a utilização destes para estabelecer um canal para controlar o fluxo de multimédia.

Descrição da média, se presente:

m = (nome da média e endereço de transporte)
i =* (título da média)
c =* (ligação informação - opcional se incluído na sessão de nível)
b =* (informação da largura de banda)
k =* (chave criptográfica)
a =* (zero ou mais linhas de atributos media)

O SDP, mais do que um protocolo, é uma forma de formatação de dados. O leitor pode ver no anexo A.1 Exemplo de SDP que foi usado no terminal de TV.

3.1.2 SDPng

O *Session Description Protocol Next Generation* (SDPng) [11], trata-se do sucessor para o SDP, é um melhoramento do seu antecessor. Baseado num código XML comum, que fornece uma biblioteca independente da aplicação para a descrição da sessão e a capacidade de descrição para diferentes cenários de aplicação.

Permite a distribuição de configurações de descrição "fixas" na difusão de cenários de aplicação, mas também suporta uma negociação dinâmica de parâmetros de sessão para conferências multimédia interativas. A fim de suportar uma ampla gama de diferentes aplicações, SDPng é completamente agnóstico, ou seja, independente da aplicação. A especificação base não define nenhum vocabulário específico para a aplicação de, por exemplo, tipos de média, *CoDecs* e os seus parâmetros de configuração, etc.., mas destina-se a ser uma biblioteca extensível que fornece mecanismos necessários de extensibilidade para suportar aplicações futuras bem como futuros mecanismos de transporte e codificação.

SDPng fornece mecanismos fundamentais que suportam a inclusão de meta-informação: a descrição dos componentes da aplicação (sessões de multimédia) e configurações alternativas que podem ser etiquetadas para ativar futuras referências, por exemplo, para a associação de meta-informação. Numa sessão em que a emissão de TV seja multilingue, a informação do idioma pode ser fornecida por um fragmento de meta-informação atribuída à *tag* <lingua> para a descrição de sessão de média referenciando-a.

SDPng, por si só, não define vocabulário para especificar meta-informação, mas permite a inclusão arbitrária de fragmentos de meta-informação, por exemplo, o MPEG-7. Estas descrições podem ser incluídas ou referenciadas por URLs.

Uma descrição SDPng é um código XML dividido no máximo em cinco partes:

• Capacidades: esta secção fornece uma lista das capacidades individuais. É opcional. O tipo de elemento é chamado de "cap".

• Definições: esta secção apresenta definições de parâmetros frequentemente utilizados para posterior referência. É opcional. O tipo de elemento é chamado de "def".

• Configurações: esta secção apresenta a descrição dos diferentes componentes de conferência (aplicações em conferência). Tem de estar presente. O tipo de elemento é chamado de "cfg".

• Restrições: esta secção apresenta constrangimentos nas combinações de configurações. É opcional. O tipo de elemento é chamado de "constraints".

• Sessão de informação: esta secção fornece meta-informações nas conferências e nos componentes individuais. É opcional. O tipo de elemento é chamado de "info".

Como referido, um SDPng é um código XML. No código, o elemento raiz deve ser do tipo "sdpng". O vocabulário XML para a especificação base do SDPng reside no XML *namespace*[2].

Um exemplo das capacidades do elemento SDPng :

```
<?xml version="1.0"?>
<sdpng xmlns="http://www.iana.org/sdpng"
      xmlns:sdpng="http://www.iana.org/sdpng">
  <cap>
     <video:codec name="h263+-enhanced">
     <video:encoding>H.263+</video:encoding>
     <video:resolution>QCIF</video:resolution>
     <video:imagemrate max="30"/>
     <h263plus:A>foo</h263plus:A>
```

[2] http://www.iana.org/sdpng

```
        <h263plus:B>bar</h263plus:B>
      </video:codec>
      [...]
    </cap>
  </sdpng>
```

O seguinte exemplo descreve um elemento do tipo “def” com um tipo de elemento de definição "RTP:UDP". Este elemento é utilizado para especificar parâmetros fixos de uma sessão RTP - os parâmetros autorizados terão sido especificados no correspondente pacote SDPng RTP. Um elemento de definição também está presente e uma sessão de informação como um ficheiro SDP.

```
  <?xml version="1.0"?>
  <sdpng xmlns="http://www.iana.org/sdpng"
        xmlns:sdpng="http://www.iana.org/sdpng">
    <cap>
      <audio:codec name="avp:pcmu">[...]</audio:codec>
      <rtp:udp name="rtpudpip6">[...]</rtp:udp>
    </cap>
    <def>
     <rtp:udp name="rtp-cfg1" ref="rtp:rtpudpip6">
       <rtp:ip-addr>::1</rtp:ip-addr>
       <rtp:rtp-port>9456</rtp:rtp-port>
       <rtp:pt>1</rtp:pt>
     </rtp:udp>
    </def>
    <cfg>
      <component name="interactive-audio" media="audio" status="active">
       <alt name"alt1">
         <audio:codec ref="avp:pcmu"/>
         <rtp:udp ref="rtp-cfg1"/>
       </alt>
     </component>
    </cfg>
    <constraints>
     [...]
    </constraints>
   <info>
      <part type="SDP">
       o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
       s=SDP SUIT
       i=Example
       t=2873397496 2873404696
      </part>
    </info>
</sdpng>
```

Após a apresentação, em geral, dos protocolos SDPng e SDP, a seguinte Tabela 1 apresenta uma breve comparação entre os dois protocolos.

	SDPng	**SDP**
Estado do documento	IETF Draft	RFC. RTP RFCs [7]
	Independente da aplicação	Depende da aplicação
Estrutura	Baseado em XML	Baseado em texto, par chave/valor
Extensibilidade	Simples. Suportes nativos para extensão	Complicado. Usa atributos e conceitos para a sua extensão
Formato metadata	Permite descrições MPEG-7 / MPEG-21 DIA	Não suporta MPEG-7/MPEG-21
Anúncios	Suporta SAP e RTSP	Suporta SAP e RTSP
Uso	Não é usado frequentemente	É usado frequentemente
Suporte	Não existe suporte	Bom suporte
MDC/SVC suporte	Ainda não foram definidas	Existem extensões definidas para MDC/SVC
Informação da sessão	Suporta SDP (informação da sessão)	-

Tabela 1 - Comparação SDPng e SDP.

Neste trabalho, usámos o protocolo SDP para o serviço VoD.

3.1.3 Descoberta do serviço e seleção (SD&S)

O padrão DVB-IP [12] define os mecanismos para descoberta dos serviços disponíveis na rede IP numa alternativa os descritos na norma DVB-PSI/SI [13]. A descoberta resulta na apresentação de uma lista de serviços para o utilizador final, usualmente denominado *Menu*, com informações suficientes para que o utilizador escolha e aceda aos serviços desejados fornecidos pelo operador.

Os serviços definidos na norma DVB-IP são:

- LMB (*Live Media Broadcast*): serviço de transmissão em direto.
- VoD (*Video on Demand*): serviço de vídeo a pedido.

Por sua vez, o serviço LMB pode ser subdividido em dois tipos:

- TS Full SI: com DVB-SI [14] integrado no fluxo. Este tipo é adequado para o caso em que o operador de serviço transmite ao utilizador final, através da rede IP, fluxos de TV digital DVB. Neste caso, o mínimo de informação que o operador de

serviço precisa gerar especificamente para transmissão em IP é a necessária para o terminal de TV localizar os fluxos de transporte. Em seguida, informações sobre serviços individuais são obtidas do próprio fluxo de transporte, através da utilização clássica, tal como definida no DVB-SI.

- TS Optional SI: sem DVB-SI integrado no fluxo, exceto as tabelas PSI do sistema MPEG-2. Este tipo é adequado quando o operador de serviço não quer usar largura de banda para transmitir as informações do DVB-SI. Neste caso, o processo de descoberta do serviço precisa fornecer ao terminal de TV a localização do serviço bem como informações relevantes sobre cada serviço oferecido.

3.1.3.1 Identificação do serviço

Um operador de serviços é identificado exclusivamente pelo seu nome do domínio (DNS). Os serviços oferecidos são identificados, apenas, através de uma das seguintes formas:

- um conjunto de identificadores numéricos (*original_network_id*, *transport_stream_id*, *service_id*), conforme definido no DVB-SI;
- um identificador textual da seguinte forma :

  <nome_do_servico>.<nome_do_dominio_do_operador>, ou seja, formado pela concatenação de um nome de serviço (gerido unicamente pelo operador) com o nome do domínio do operador, conforme definido no MHP[15].

3.1.3.1 Registos de SD&S

As informações para descoberta e seleção dos serviços (SD&S - *Service Discovery and Selection*) são representadas e transportadas em código XML. A estrutura do XML está representado no anexo C de [12].

As informações de SD&S são classificadas em 6 tipos, com possibilidade de extensão futura.

De acordo com o seu tipo, as informações de SD&S referem-se:

- a um operador de serviço;
- a um serviço de transmissão em direto;
- a um serviço de vídeo a pedido;
- a serviços oferecidos por outros operadores;
- a um pacote de serviços agrupados;
- a um serviço de guia de programação (EPG).

O tipo de informação de SD&S é identificado pelo campo *Payload ID* de 8 bits, conforme

especificado na Tabela 2.

Payload ID	**Registo SD&S**
0x00	Reservado
0x01	Operador de serviço
0x02	Serviço de transmissão ao vivo
0x03	Serviço de vídeo a pedido
0x04	Serviços de outros operadores
0x05	Pacote de serviços
0x06	BCG
0x07 – 0xEF	Reservado
0xF0 – 0xFF	Uso privado

Tabela 2 -Valores de *Payload ID* dos registos de SD&S.

Os códigos XML de SD&S podem ter um tamanho relativamente grande. Porém apenas uma parte deles é necessária ao terminal de TV num determinado instante. Além disso, as alterações nesses registos podem ser localizadas numa parte apenas. Por essas razões, os registos SD&S podem ser segmentados em unidades menores. Os segmentos são definidos no contexto de um único tipo de informação de SD&S, ou seja, para um dado *Payload ID*.

Cada segmento é um código XML válido e é identificado por um identificador (*Segment ID*) de 16 bits. As alterações nos segmentos são identificadas pelo campo da versão do segmento, de 8 bits. Apenas e só, quando um dado do segmento sofre uma alteração, a versão deve ser incrementada, voltando a zero após 255.

O tempo de ciclo é definido pelo intervalo de tempo requerido para transmitir todos os segmentos que compõem os registos SD&S de um operador. O tempo máximo de ciclo é de 30 segundos.

A Figura 12 ilustra o relacionamento entre registos e segmentos de SD&S.

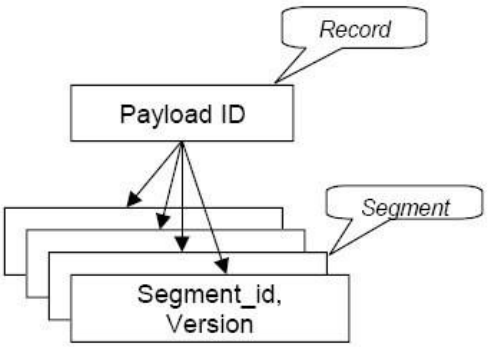


Figura 12 - Relacionamento entre registos e segmentos de SD&S.

Os registos de SD&S podem ser transmitidos através de *multicast* (modo *push*) ou obtidos por pedido (modo *pull*). A mesma informação pode ser transmitida em ambos os modos. Os mecanismos de transporte dos registos de SD&S serão apresentados, posteriormente, na secção 3.1.3.8.

3.1.3.3 Passos na descoberta do serviço

O processo de descoberta dos serviços possui os seguintes passos:

1. Encontrar os pontos de entrada para a descoberta dos serviços;
2. Em cada ponto de entrada, descobrir os operadores de serviço;
3. Para cada operador de serviço, descobrir os serviços DVB-IP disponíveis.

3.1.3.4 Pontos de entrada

Os pontos de entrada para a descoberta dos serviços podem ser um dos seguintes:

- Um endereço *multicast* conhecido, registrado no IANA: 224.0.23.14 (DbvServDisc).
- Um nome DNS conhecido, conforme o RFC 2782 [16]. O nome do serviço é _dvbservdsc, o protocolo utilizado pode ser o TCP (servidor HTTP) ou UDP (endereços multicast), e o restante do nome é o domínio mantido pelo DVB para a descoberta do serviço: services.dvb.org. Desta forma, a busca pode ser feita em _dvbservdsc._tcp.services.dvb.org ou _dvbservdsc._udp.services.dvb.org. Isto requer que o terminal de TV possua um cliente DNS compatível com o RFC 2782. Os novos operadores de serviço devem registar-se no DVB para que sejam incluídos na lista DNS SRV.
- Um nome DNS semelhante ao item anterior, contudo o nome de domínio será fornecido pelo DHCP ao terminal de TV durante a sua inicialização. Por exemplo, a busca seria feita em _dvbservdsc._tcp.suit.org, onde o domínio suit.org foi fornecido pelo DHCP.
- Endereços fornecidos na configuração do terminal de TV através da rede, quando esta funcionalidade estiver implementada.

Cada ponto, descrito em cima, possui precedência sobre o item anterior. Se nenhum ponto de entrada for obtido através das alternativas descritas em cima, deverá haver uma opção para o utilizador introduzir manualmente o URL ou o endereço IP, com o número da porta

opcional, do ponto de entrada. Em todos os casos, caso a porta não esteja definida, deverá ser utilizada por defeito a porta 3937.

3.1.3.5 Informação de descoberta de operadores de serviço

As principais informações contidas nos registos de SD&S relativos aos operadores de serviço são as seguintes:

- Nome do domínio DNS do operador de serviço.
- Versão do registo, que deve ser incrementada sempre que houver alteração de alguma informação no código XML.
- Nome do operador de serviço, em um ou mais idiomas, para mostrar ao utilizador.
- Descrição (opcional) do operador de serviço, em um ou mais idiomas, para mostrar ao utilizador.
- Localização (URL) dos registos de SD&S relativos aos serviços de DVB-IP oferecidos pelo operador.
- *Payload Id*: Indica o tipo de informação de descoberta de serviço disponível na URL especificada no campo de localização, conforme definido na Tabela X.
- Lista com o identificador dos segmentos que contêm a informação de descoberta de serviço do tipo definido no *Payload Id*, e as suas respetivas versões. É obrigatório caso o registo de SD&S seja transmitido por *unicast*, e opcional caso seja por *multicast* (maios detalhes da transmissão dos registos de SD&S serão apresentados na secção 3.1.3.8).

A lista completa dos campos dos registos de SD&S relativos aos operadores de serviço DVB-IP está especificada em [12].

3.1.3.6 Informação de descoberta dos serviços

Os registos de SD&S relativos aos serviços DVB-IP contêm as seguintes informações comuns:

- Nome do domínio DNS do operador de serviço.
- Versão do registo, que deve ser incrementado sempre que houver alteração de alguma informação no código XML

Além dessas informações comuns, os registos de SD&S relativos aos serviços contêm informações específicas de acordo com o tipo de serviço oferecido: LMB TS *Full* SI, LMB TS Optional SI ou VoD. O operador do serviço também pode fazer referência a serviços

fornecidos por outro operador. Neste caso, as informações sobre esses serviços, como a sua localização, precisam de ser obtidas diretamente do operador que os fornece. O operador também pode definir um pacote de serviços, para apresentá-los de forma agrupada.

O registo de SD&S do serviço LMB TS *Full* SI fornece as informações necessárias para a localização dos serviços disponíveis de emissão em direto com SI incorporado. Informações específicas dos serviços individuais são obtidas posteriormente a partir do próprio fluxo de transporte, através do uso clássico do DVB-SI. Além das informações comuns apresentadas, as principais informações contidas neste registo são, para cada serviço LMB TS Full SI oferecido:

- Nome DNS do *host*, único para o serviço, no domínio controlado pelo operador.
- Lista de objetos de identificação do serviço DVB, conforme apresentado anteriormente, secção 3.1.3.1.
- Localização do serviço, que pode ser um endereço IP *multicast* ou uma URL de um servidor RTSP.

O registo de SD&S do serviço LMB TS opcional SI fornece todas as informações necessárias para criar uma lista de serviços disponíveis, de modo a que o utilizador possa fazer a sua escolha e aceder ao serviço. As principais informações específicas contidas neste registo são, para cada serviço LMB TS opcional SI oferecido.

- Nome DNS do *host*, único para o serviço, no domínio controlado pelo operador de serviço.
- Lista dos objetos de identificação do serviço DVB, conforme apresentado anteriormente, secção 3.1.3.1.
- Nome do serviço, em um ou mais idiomas, para mostrar ao utilizador.
- (Opcional) Descrição do serviço, em um ou mais idiomas, para mostrar ao utilizador.
- (Opcional) Descrição do conteúdo deste serviço, indicando o tipo de programação. Por exemplo: filme de drama, notícias, musica, etc.
- Tipo de serviço DVB, conforme especificado em [14]. Por exemplo: TV digital, radio digital, transmissão de dados, DVB MHP, etc.
- (Opcional) O elemento de suporte a anúncios, que identifica os tipos de anúncios falados que são suportados pelo serviço. Por exemplo: chamada de emergência, informação de trânsito, meteorologia, etc. Além disso, informa o método de transporte do anúncio.
- (Opcional) Descrição de serviço alternativo, selecionado automaticamente no caso de falha da descodificação do serviço atual.

- (Opcional) Fonte primária de informação do serviço, que indica qual fonte de informação do serviço que tem prioridade (código XML ou DVB SI), no caso das tabelas DVB SI estarem presentes. O por defeito, código XML é prioritário.
- Localização do serviço, que pode ser um endereço IP *multicast* ou uma URL de um servidor RTSP.

O registo de SD&S do serviço VoD fornece todas as informações necessárias para a descoberta dos servidores de VoD disponíveis na rede e a localização da sua lista de conteúdos. Não fornece nenhuma informação sobre os conteúdos individuais. As informações específicas contidas neste registo são, para cada operador/servidor de VoD:

- Identificador do operador/servidor de VoD. Este identificador é alocado pelo operador.
- Nome da lista de serviços de VoD disponíveis, em um ou mais idiomas, para mostrar ao utilizador.
- (Opcional) Descrição geral dos serviços de VoD disponíveis, em um ou mais idiomas, para mostrar ao utilizador.
- Localização (URL) da lista de serviços de VoD disponíveis. Um pedido HTTP a esta URL deve devolver um registo compatível com um esquema que vai ser especificado numa nova atualização da especificação do DVB-IP.

Um operador pode ainda fazer referência a serviços individuais ou à oferta completa de outro operador. Estas informações estão contidas num outro tipo de registo de SD&S, denominado "registo de SD&S de serviços de outros operadores". As informações específicas contidas neste registo são, para cada operador referenciado:

- Nome do domínio DNS do operador referenciado.
- (Opcional) Para cada serviço referenciado, o nome DNS do *host*, único para o serviço, no domínio controlado pelo operador referenciado. Caso não seja especificado nenhum serviço, o registo faz referência à oferta completa de serviços do operador referenciado.

O último tipo de registo de SD&S especificado no padrão DVB-IP é o registo de descoberta de um pacote de serviços, ou seja, um conjunto de serviços oferecidos por um operador e associados numa única entidade. Um dado serviço pode pertencer a um ou mais pacotes, ou não pertencer a nenhum pacote. As principais informações específicas contidas neste registo são, para cada pacote de serviços:

- Identificador do pacote, alocado pelo operador.
- Nome do pacote, num ou mais idiomas, para mostrar ao utilizador.
- Lista dos serviços que compõem o pacote, com seus respetivos nomes DNS do *host*, no domínio controlado pelo operador.

A lista completa dos campos dos registos de SD&S relativos aos diversos serviços DVB-IP está especificada em [12].

3.1.3.7 Seleção do serviço

Um serviço pode ser acedido por um terminal de TV da seguinte forma:

- RTSP
- IGMP

Serviços de emissão em direto (LMB) são transmitidos sobre IP *multicast*. Os fluxos de transmissão são contínuos e não precisam de ser iniciados por cada terminal de TV. Os terminais de TV podem entrar e sair do grupo *multicast*, relativo a um serviço LMB, através do envio de mensagens IGMP JOIN e LEAVE, respetivamente, será abordado com mais detalhe na secção 3.2.1. O atributo de localização do serviço, presente no registo de descoberta do serviço, fornece a informação necessária para a geração das mensagens IGMP apropriadas. Nenhum controlo do fluxo de transmissão nestas condições, por exemplo, fazer uma pausa ou avançar.

Opcionalmente, o operador pode exigir que o terminal de TV que execute explicitamente as fases de inicialização e finalização do serviço LMB. Um das razões possíveis para que isso aconteça é a cobrança de um serviço, acesso condicional, entre outas. Neste caso, o atributo de localização do serviço, presente no registo de descoberta do serviço, informa que deve ser utilizado o protocolo RTSP, e fornece a informação necessária para o acesso por esse protocolo. Os parâmetros necessários para a geração da mensagem IGMP são obtidos através do método SETUP do RTSP. Na secção 3.2.2 será abordado, com mais detalhes, a utilização do RTSP no terminal de TV.

3.1.3.8 Mecanismos de transporte dos registos de SD&S

Os registos de SD&S de descoberta dos operadores e dos serviços podem ser transmitidos por *multicast* ou por *unicast*.

3.1.3.9 Multicast

O DVB define um novo protocolo para a transmissão do código XML com informações de SD&S sobre pacotes UDP *multicast*. Este protocolo é denominado DVB SD&S *Transport Protocol* (DVBSTP) [12], e o formato do pacote está representado na Figura 13.

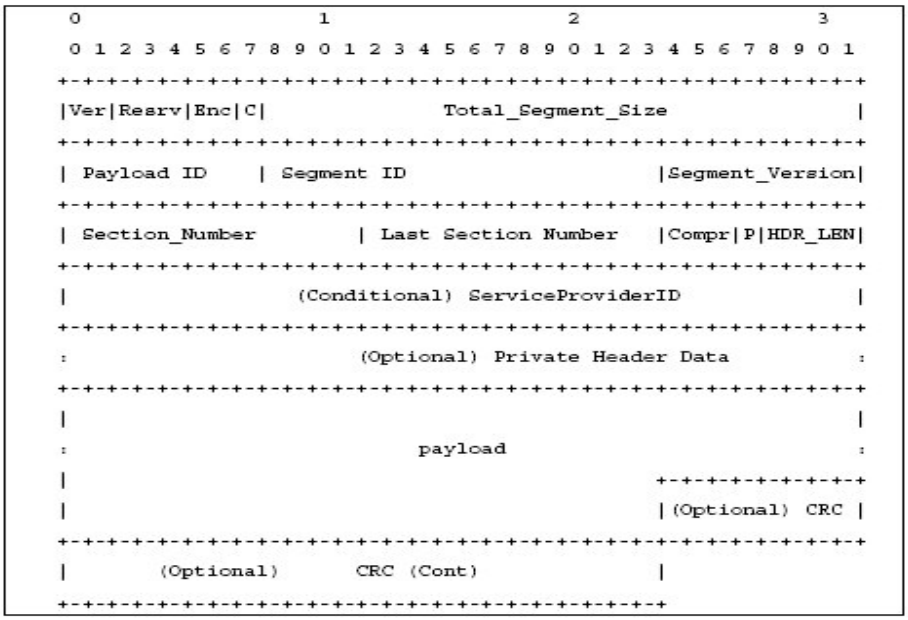


Figura 13 - Formato do pacote DVBSTP.

Descrição dos campos do pacote DVBSTP:

- Version (Ver): Versão do protocolo. Campo de 2 bits deve ter o valor “00”.
- Reserved (Resrv): Reservado. Campo de 3 bits deve ter o valor “000”.
- Encription (Enc): Campo de 2 bits deve ser usado para indicar a presença de criptografia. De momento só está definido o valor “00”, que indica a ausência de criptografia no *payload*.
- CRC flag (C): O valor “1” indica a presença de um campo de CRC de 32 bits no final do pacote. Isto só pode ocorrer no último pacote do segmento, ou seja, quando o valor do campo *Section Number* for igual ao valor do campo *Last Section Number*.
- Total Segment Size: Campo de 24 bits indica um tamanho em bytes. Para dados descompactados (isto é, compressão com o valor "000"), ou para dados compactados que podem ser utilizados na forma compacta, é o tamanho acumulado do *payload* de todas as secções que formam o segmento (isto é, ignorar os cabeçalhos e CRC, se presente). Para dados compactados que precisam de ser

descompactados antes de serem utilizados, é o tamanho do segmento depois de descompactado.

- Payload ID: Campo de 8 bits. Identificador do tipo de dados contidos no *payload*. Os valores possíveis estão especificados na Tabela 2.
- Segment ID: Campo de 16 bits usado para identificar o segmento de dados para o tipo de *Payload ID* declarado. Por exemplo, pode haver múltiplos registos de descoberta de serviços de transmissão em direto, e cada um terá um identificador de segmento único.
- Segment Version: Campo de 8 bits usado para definir a versão actual do segmento. Quando algum dado dentro do segmento se altera, os campos de versão do segmento de todos os pacotes com este *Segment ID* e *Payload ID* são incrementados, voltando a zero após 255.
- Section Number: Campo de 12 bits que identifica o número da secção atual. A primeira secção do segmento deve ter valor zero.
- Last Section Number: Campo de 12 bits que identifica o número da última secção do segmento.
- Compression (Compr): Campo de 3 bits usado para indicar o tipo de compressão dos dados do *payload*, se houver. Todos os segmentos de um dado *Payload ID* devem possuir o mesmo tipo de compressão. O valor "000" indica que não há compressão.
- Provider ID Flag (P): O valor "1" indica a presença do campo de identificação do operador no cabeçalho.
- Private Header Length (HDR_LEN): Campo de 4 bits que indica o número de sequências de 32 bits no cabeçalho imediatamente após este campo, ou após o campo *Service Provider ID*, se este estiver presente. É usado para indicar a presença de dados privados no cabeçalho.
- Service Provider ID: Um número de 32 bits que é usado como identificador do operador, para uso do terminal de TV. Este campo deve ser um endereço IPv4. Este campo é obrigatório caso mais de um operador possa utilizar o mesmo endereço *multicast*.
- Private Header Data: São dados privados. O significado, a sintaxe, a semântica e o uso deste campo estão fora das especificações do padrão DVB-IP. O tamanho deve ser múltiplo de 4 bytes.
- Payload: Carga útil do pacote, que deve ter um número inteiro de bytes.
- CRC: Campo opcional de 32 bits para deteção de erros. O CRC deve ser calculado sobre o *payload* de todas as secções que compõem o segmento.

A subdivisão do segmento em secções é motivada pelo fato de o tamanho dos segmentos poder ser maior que o tamanho suportado pelas camadas inferiores da rede. A utilização de secções permite uma transmissão mais eficiente dos pacotes DVBSTP. Cada secção deve ser enviada em, exatamente, um datagrama UDP, e cada datagrama UDP deve conter exatamente uma secção.

Para construir o segmento inteiro, o terminal de TV obtém o *payload* de todas as secções e ordena com base no número da secção (campo *Section Number*). Após a construção do segmento, o CRC pode ser determinado e verificado, se estiver presente.

A figura 14 ilustra a relação entre registos, segmentos e secções.

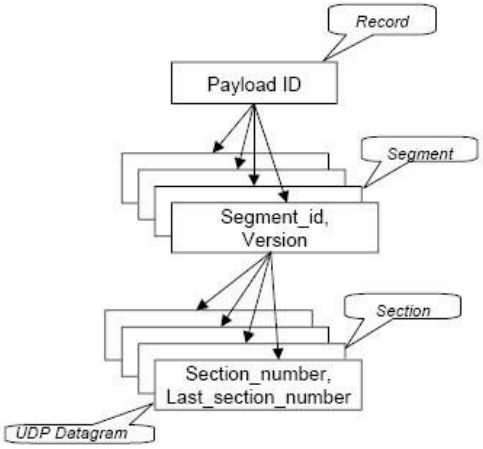


Figura 14 - Relação entre registos, segmentos e secções.

O tamanho de uma secção é limitado pelo tamanho máximo de um datagrama IP (65535 octetos para IPv4), subtraindo os tamanhos dos cabeçalhos UDP e do protocolo *multicast*. Para evitar a fragmentação na rede, o tamanho máximo da secção não pode exceder o MTU (*Maximum Transmission Unit*) da rede. No caso de uma rede *Ethernet*, que tem MTU de 1492, o tamanho da secção deve ser limitado a 1452 bytes. Se forem usadas opções adicionais dos protocolos IP, UDP ou *multicast*, este valor deve ser reduzido pela quantidade apropriada.

Todos os segmentos dos registos de SD&S de um operador devem ser transmitidos em intervalos de tempo não superiores ao ciclo de tempo máximo, que é de 30 segundos, conforme foi visto anteriormente. Um segmento pode ser transmitido várias vezes durante este tempo, e segmentos diferentes podem ser transmitidos com taxas diferentes.

3.1.3.10 Unicast

Para o transporte de informações de SD&S em *unicast*, deve ser utilizado o protocolo HTTP.
De modo a solicitar informações de SD&S, o terminal de TV deve enviar uma mensagem HTTP com o seguinte formato:

```
'GET /dvb/sdns' request ' HTTP/1.1' CRLF
'Host: ' host CRLF
```

No caso de uma solicitação de informação de descoberta relativa a operadores, request = sp_discovery, o *host* é o endereço IP do ponto de entrada das informações de SD&S, obtido conforme foi apresentado anteriormente secção 3.1.3.4. O pedido possui um parâmetro id, que pode ter o valor "*ALL*", quando o pedido refere-se a todos os operadores, ou o nome do domínio de um operador específico, quando o pedido refere-se somente a ele. Portanto, há duas formas de efetuar o pedido HTTP para a obtenção da informação sobre os operadores:

```
'GET /dvb/sdns/sp_discovery?id=ALL HTTP/1.1' CRLF
'Host: ' host CRLF

'GET /dvb/sdns/sp_discovery?id=' DomainName ' HTTP/1.1' CRLF
'Host: ' host CRLF
```

A resposta ao pedido HTTP deve devolver um código XML de SD&S com informações da descoberta dos serviços oferecidos pelos operadores, conforme apresentado anteriormente secção 3.1.3.5. Numa emissão *unicast*, esses registos não podem ser fragmentados. No caso de um pedido de informação de descoberta relativa à oferta de serviços de um operador, request = service_discovery, e o *host* é o endereço especificado no registo de SD&S do operador, no campo que informa a localização dos registos de SD&S relativos aos seus serviços, conforme apresentado anteriormente secção 3.1.3.5. O pedido possui três parâmetros obrigatórios: nome do domínio do operador, tipo de serviço (*Payload ID*) e identificação do segmento (*Segment ID*). Todos esses parâmetros são obtidos a partir do registo de SD&S do operador. Opcionalmente, o pedido pode ter um parâmetro que informa a versão do segmento que o terminal de TV possui no momento, de modo a que o servidor só devolva o registo caso haja uma nova versão disponível.
Por exemplo, o pedido descrito, refere-se à oferta de serviços de transmissão em direto de um operador com identificador suit.org:

```
'GET /dvb/sdns/service_discovery?id=suit.org&Payload=02&Segment=0001
HTTP/1.1' CRLF
'Host: ' host CRLF
```

A resposta ao pedido HTTP sobre a oferta de serviços de um operador deve devolver o

código XML do SD&S com informações de descoberta de serviços, conforme apresentado anteriormente na secção 3.1.3.6., podemos ver um exemplo desse código XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<dvb:ServiceDiscovery xmlns:dvb="urn:dvb:ipi:sdns:2006>
<dvb:ServiceProviderDiscovery>
      <dvb:ServiceProvider DomainName="suit.org" Version="0">
            <dvb:Name Language="eng">SUIT</dvb:Name>
            <dvb:Description Language="eng">
                  The SUIT project offers Live TV ...
            </dvb:Description>
            <dvb:Offering>
                  <dvb:Pull Location="http://x.x.x.x:3937/dvb/sdns">
                        <dvb:PayloadId Id="2">
                              <dvb:Segment ID="0" Version="0"/>
                        </dvb:PayloadId>
                  </dvb:Pull>
                  <dvb:Push Port="3937" Address="224.0.23.14">
                        <dvb:PayloadId Id="5"/>
                  </dvb:Push>
            </dvb:Offering>
      </dvb:ServiceProvider>
</dvb:ServiceProviderDiscovery>
</dvb:ServiceDiscovery>
```

O terminal de TV deve obter periodicamente o registo de SD&S relativo ao operador, para identificar uma possível atualização da versão. Caso a versão tenha sido alterada, o terminal de TV deve verificar a lista de identificadores de segmentos, para identificar qual segmento que teve a versão alterada, e obter novamente o registo de SD&S correspondente.

3.2 Protocolos de gestão de sessão

3.2.1 IGMP

O *Internet Group Management Protocol* (IGMP) [17],[18], permite aos terminais juntarem-se e abandonarem grupos *multicast*. Enviando um relatório de associação ou parceria (*membership report*) ao *router* de vizinhança imediata, uma estação informa o *router* que deseja fazer parte de um grupo *multicast*. Os *routers* transmitem periodicamente mensagens com interrogações de associação (*membership query*) para determinar quais os "*host groups*" que têm membros nas suas redes diretamente ligados.

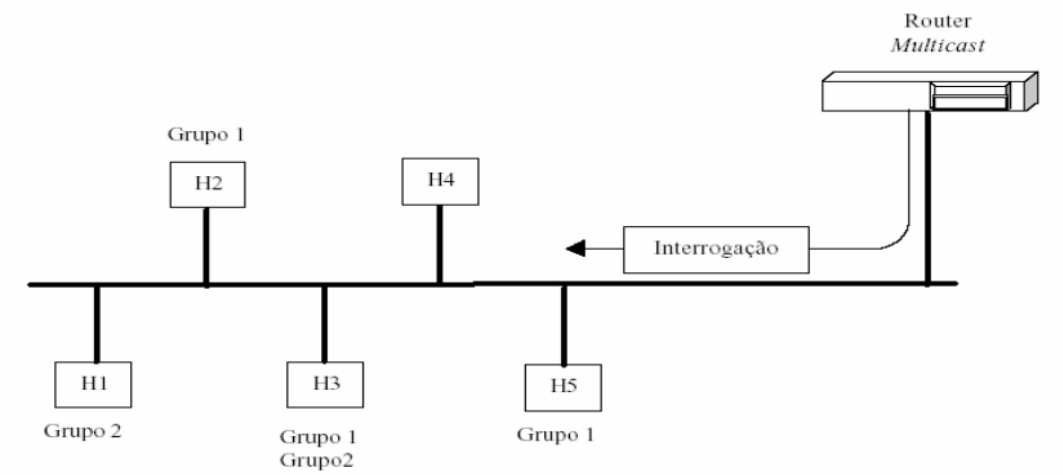


Figura 15 - IGMP Mensagem de interrogação.

A versão 1 deste protocolo, o IGMPv1, está especificada no RFC-1112, Apêndice 1. Essas mensagens de interrogação IGMP (ver Figura 15) são endereçadas a todos os *hosts* (224.0.0.1) e possuem um TTL (*Time To Live*) IP de 1 para limitar a sua transmissão apenas na sub-rede onde tiveram origem e não são encaminhadas por qualquer outro *router multicast*. Um *host* responde com um *membership report* para cada grupo ao qual pertence. Para limitar o número *membership reports*, cada estação inicia uma espera de tempo aleatório depois de ter recebido o *membership query*. As estações "ouvem" o meio tomando conhecimento dos relatórios de parceria enviados ao *router*; se um relatório é submetido para o grupo ao qual a estação pertence o seu tempo de espera expira, e cancela o seu relatório para o grupo. Através deste mecanismo apenas um *membership report* é gerado por cada grupo. Baseado nas informações que estão estabelecidas nos grupos fornecidas através do IGMP, os *routers* têm a capacidade para determinar que tráfego *multicast* (se houver algum) se deve encaminhar para as redes interligadas.

Quando o *software* aplicacional pede ao software de rede da estação para esta se juntar a um grupo *multicast*, uma mensagem IGMP é enviada ao *router* mais próximo (se o *host* não for já um membro do grupo). Ao mesmo tempo, o endereço *multicast* de classe D do grupo ao qual se junta é mapeado como um endereço de baixo nível e a interface da rede é programada para aceitar pacotes para esse endereço. Por exemplo, se uma estação passa a integrar um grupo numa interface *Ethernet*, os 23 bits mais baixos do endereço de classe D são mapeados aos 23 bits mais baixo do endereço *Ethernet*. Resultante desta filtragem de endereços *multicast* por *hardware*, um *router* não necessita de manter uma lista detalhada das estações que pertencem a cada endereço de grupo, mas apenas esse membro, pelo menos, do grupo, está presente na sub-rede à qual se encontra vinculado. Uma das debilidades da primeira versão do IGMP era a latência demasiado elevada associada com a terminação das sessões *multicast*. Depois do último membro de um grupo *multicast* numa sub-rede ter abandonado o grupo, os outros *routers* não são imediatamente notificados para parar a propagação de tráfego para o grupo. Esta demora era causada pelo IGMP tendo que

aguardar pelas várias interrogações que indicassem que não havia mais membros na sub-rede, de um grupo em particular. No entretanto, indesejavelmente, o tráfego desnecessário seria encaminhado para a sub-rede. O "custo" deste envio inútil podia ser elevado, particularmente num segmento da *internet* com largura de banda limitada.

A versão 2 do IGMP [17], apresenta alguns melhoramentos que ajudam a reduzir o *overhead* do protocolo. As mensagens de interrogação dirigidas a grupos específicos (*Group Specific Query Message*) permitem ao *router* interrogar grupos específicos nas redes onde estão diretamente vinculados em vez de serem forçados a interrogar todos os grupos indiscriminadamente. Começando com a versão 2, a terminação de uma sessão *multicast* já não é feito de forma passiva. O último *host* de uma sub-rede a deixar o grupo *multicast*, transmite uma mensagem de saída de grupo (*Leave Group*) ao *router* onde é indicado qual o grupo abandonado. Depois de verificar a partida com uma mensagem de interrogação dirigida a esse grupo específico, o *router* notifica outros *routers* para finalizarem o encaminhamento de tráfego para a sub-rede dirigida ao grupo.

A versão 3 do IGMP [18] vai mais longe na redução do *overhead*. A largura de banda será conservada pela mensagem *Group-Source Report* que permitirá às estações receber tráfego de fontes específicas de um grupo *multicast*. Em versões anteriores do IGMP, o tráfego de todas as fontes tinha de ser encaminhado para uma sub-rede mesmo se as estações estivessem apenas interessadas em receber tráfego de fontes específicas. As mensagens *Leave-Group* apresentadas em primeira instância pela versão 2 foram também aperfeiçoadas para permitir às estações largar um grupo inteiro ou para especificar a fonte a que queriam renunciar.

Atendendo a que as versões recentes do IGMP podem reduzir o tráfego desnecessário, otimizando a utilização deste protocolo, deve ser favorecida a sua utilização em detrimento das anteriores. Pelos métodos acima mencionados, os *routers multicast* estão habilitados a manter, por interface, uma tabela atualizada contendo os grupos cujo tráfego tem interesse para a sub-rede, pelo que após a receção de pacotes *multicast*, os *mrouters* sabem para que interfaces os pacotes devem ser encaminhados.

O IGMP é utilizado para trocar informações sobre o estado dos membros do grupo entre os *routers* IP que suportam *multicast* e os membros dos grupos *multicast*. A adesão de *hosts* a um grupo *multicast* é relatada por *hosts* membros individuais e o estado de adesão é apurado periodicamente por *routers multicast*, como descrito anteriormente.

Os tipos de mensagem IGMP são descritos na tabela a seguir.

Tipo de mensagem IGMP	Descrição
Relatório de membros do grupo de anfitriões	Enviado quando um *host* adere a um grupo *multicast* para declarar a entrada de um membro num determinado grupo de anfitriões. As mensagens de relatório de membro do grupo *host* IGMP também são enviadas em resposta a uma consulta de associação a um grupo *host* IGMP enviada por um *router*. Para uma mensagem de relatório de membro *host* do IGMP versão 3, o *host* pode especificar o interesse em receber tráfego *multicast* de origens especificadas ou de todas menos de um determinado conjunto de origens. O relato específico da origem evita que os *routers* ativados através de *multicast* entreguem o tráfego *multicast* numa sub-rede onde não existam *hosts* em escuta.
Consulta aos membros do grupo de anfitriões	Utilizada por um *router multicast* para apurar periodicamente os membros dos grupos numa rede. Para uma mensagem de consulta aos membros anfitriões do IGMP versão 3, o *router* pode consultar o interesse do *host* em receber tráfego *multicast* de uma lista de origens especificada.
Abandonar grupo	Enviada por um *host* quando abandona um grupo de anfitriões e é o último membro do respetivo grupo no segmento de rede.

Tabela 3 – Tipo de mensagem IGMP

As mensagens IGMP são encapsuladas e enviadas em datagramas IP, conforme apresentado na figura seguinte, Figura 16.

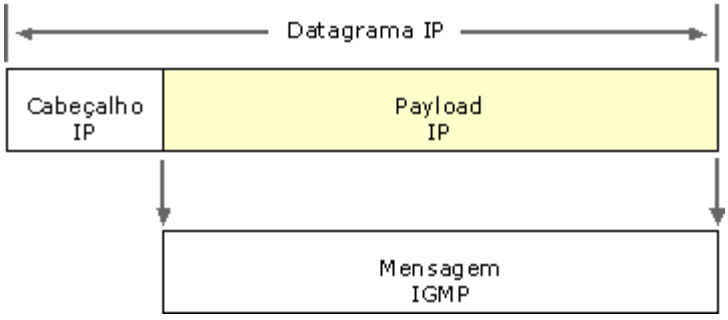


Figura 16 - Datagrama IP.

3.2.2 RTSP

O RTSP [10], é um protocolo não orientado à ligação, para controlar a transmissão dos fluxos em direto e conteúdos de vídeo a pedido (VoD), o servidor mantém uma sessão associada a um identificador, na maioria dos casos o RTSP usa o TCP para os dados de

controlo e o UDP para os dados de áudio e vídeo, contudo também pode usar-se o TCP, caso que seja necessário.

Os dados de controlo RTSP transmitidos entre o terminal de TV e o servidor devem utilizar uma ligação TCP permanente. Isso permite que o servidor RTSP envie mensagens assíncronas a um terminal de TV que esteja protegido por uma *firewall*.

Os fluxos de dados multimídia, encapsulados em RTP, podem ser transmitidos pelo servidor RTSP tanto em modo *unicast* como em modo *multicast*. Porém, em modo *multicast* não podem ser realizadas operações de controlo da emissão, como pausa, *fast forward*, etc.

A sintaxe e a operação são semelhantes ao HTTP/1.1. No entanto, o RTSP difere do HTTP em alguns aspetos:

- O RTSP introduz novos métodos e tem um identificador de protocolo diferente.
- Um servidor RTSP precisa de manter o estado da ligação ao contrário do HTTP.
- Tanto o servidor como o cliente podem lançar pedidos.
- Os dados são transportados por um protocolo diferente.

O RTSP tem as seguintes propriedades:

- **Extensivel:** novos métodos e parâmetros podem ser facilmente acrescentados ao RTSP;

- **Seguro:** RTSP pode utilizar mecanismos de segurança por exemplo o protocolo de transporte TLS;

- **Independente do protocolo de transporte:** RTSP pode usar o protocolo UDP ou TCP;

- **Capacidade multi-servidor:** Cada fluxo de multimédia dentro de um dado serviço pode estar em servidores diferentes, o cliente automaticamente estabelece várias sessões concorrentes de controlo com os diferentes servidores:

- **Controle de dispositivos de gravação:** O protocolo pode controlar dispositivos de gravação e reprodução;

- **Adequado para aplicações profissionais:** RTSP suporta resolução a nível de imagem mediante marcas temporárias SMPTE para permitir a edição digital;

Vamos apresentar os vários passos na interação do terminal de TV com o servidor:

- Obtenção da descrição detalhada do objeto multimédia (DESCRIBE *request*)
- Início de uma sessão (SETUP *request*)
- Emissão de pedidos (PLAY, RECORD, PAUSE)
- Conclusão da sessão (TEARDOWN *request*)

a) DESCRIBE (recomendado)

Este método obtém uma descrição detalhada de uma apresentação ou do objeto multimédia indicado por uma URL RTSP que se encontra num servidor.

O servidor responde a este pedido com uma descrição do recurso solicitado, entre outros dados a descrição contém uma lista dos fluxos multimédia que serão necessários para a reprodução. Esta solicitação/resposta constitui a fase de início do RTSP.

b) SETUP (obrigatório)

• Criar um *socket* para receber os dados RTP e atribuir-lhe um *timeout* de 5 milissegundos

• Enviar pedido SETUP ao servidor. Necessita de inserir o cabeçalho *transport* no qual terá de indicar a porta do *socket* dos dados RTP que criou.

• Ler a resposta do servidor e fazer o *parsing* do cabeçalho *session* para obter o seu ID.

c) PLAY (obrigatório)

• Enviar pedido PLAY ao servidor. Deve inserir o cabeçalho *session* e utilizar o ID devolvido na resposta ao SETUP. Não se deve colocar o cabeçalho *transport* neste pedido.

• Ler a resposta do servidor

d) PAUSE (obrigatório)

• Enviar pedido PAUSE ao servidor. Deverá inserir o cabeçalho *session* e utilizar o ID da sessão devolvido na resposta ao SETUP. Também não se deve colocar o cabeçalho *transport*.

• Ler a resposta do servidor

e) TEARDOWN (obrigatório)

• Enviar pedido TEARDOWN ao servidor. Deverá inserir o cabeçalho *session* e utilizar o ID da sessão devolvido na resposta ao SETUP. Não colocar o cabeçalho *transport*.

• Ler a resposta do servidor

3.3 Protocolo do canal de retorno

3.3.1 MPEG-21 DIA UED

O MPEG-21 DIA-UED [19] vai permitir enviar uma descrição do terminal para o *playout*, ou seja, informar o *playout* quais são as suas capacidades a nível da resolução suportada, espaço em disco, nome de utilizador, nome da sessão entre outras informações que permita o *playout* ajustar a largura de banda adequada às especificações do terminal.

```
<DIA xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaLocation="urn:mpeg:mpeg21:2003:01-DIA-NS UsageEnvironment.xsd"
xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS">
 <Description xsi:type="UsageEnvironmentType">
  <UsageEnvironmentProperty xsi:type="TerminalsType">
   <Terminal>
    <TerminalCapability xsi:type="CodecCapabilitiesType">
     <Decoding xsi:type="VideoCapabilitiesType">
      <Format href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:4">
       <mpeg7:Name xml:lang="en" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001">
        MPEG-4
       </mpeg7:Name>
```

```
      </Format>
      <CodecParameter xsi:type="CodecParameterBitRateType">
       <BitRate average="5000000" maximum="20000000" />
      </CodecParameter>
     </Decoding>
    </TerminalCapability>
    <TerminalCapability xsi:type="DisplaysType">
     <Display>
      <DisplayCapability xsi:type="DisplayCapabilityType" bitsPerPixel="8"
                    colorCapable="true" activeDisplay="true">
       <Mode refreshRate="25">
        <Resolution horizontal="352" vertical="288" />
        <Resolution horizontal="176" vertical="144" />
       </Mode>
       <ScreenSize horizontal="80" vertical="50" />
       <RenderingFormat>Progressive</RenderingFormat>
      </DisplayCapability>
     </Display>
    </TerminalCapability>
    <TerminalCapability xsi:type="DeviceClassType">
     <DeviceClass href="urn:mpeg:mpeg21:2003:01-DIA-DeviceClassCS-NS:1">
      <mpeg7:Name xml:lang="en" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001">
       PDA
      </mpeg7:Name>
     </DeviceClass>
    </TerminalCapability>
    <TerminalCapability xsi:type="PowerCharacteristicsType"
     averageAmpereConsumption="2.5" batteryCapacityRemaining="5"
     batteryTimeRemaining="7200" runningOnBatteries="true" />
   </Terminal>
  </UsageEnvironmentProperty>
  <UsageEnvironmentProperty xsi:type="NetworksType">
   <Network>
    <NetworkCharacteristic xsi:type="NetworkCapabilityType"
     maxCapacity="20000000" minGuaranteed="64000" inSequenceDelivery="false"
     errorDelivery="false" errorCorrection="false" />
    <NetworkCharacteristic xsi:type="NetworkConditionType"
                    startTime="…" duration="…">
     <AvailableBandwidth average="8000000" />
     <Delay packetTwoWay="200" packetOneWay="100" delayVariation="10" />
     <Error packetLossRate="0.05" />
    </NetworkCharacteristic>
   </Network>
  </UsageEnvironmentProperty>
 </Description>
</DIA>
```

Como se pode ver pela descrição do código XML, o MPEG-21 DIA UED permite a adaptação de conteúdos multimédia às capacidades do terminal e da rede de forma a obter a melhor qualidade de serviço (QoS) possível através do envio das capacidade de descodificação e tipo do vídeo, taxa de transmissão, resolução do terminal, características da rede, entre outras.

3.4 Protocolos de Transporte

Os terminais de TV requerem mecanismos eficientes de transporte para a informação transmitida dos vários serviços oferecidos pelos operadores, informação essa que é dependente do tempo. O RTP (*Real-Time Transport Protocol*) [20], protocolo de transporte em tempo-real, foi desenvolvido para responder a esta situação. Não inclui mecanismos que garantem a entrega e com garantias de QoS, ou que permitam reserva de recursos, deixando isso para as camadas de serviços inferiores. O RTP fornece mecanismos para incluir: *timestamp*, identificando diferentes tipos de *payload*, numeração de sequenciação, e monitorização da entrega da informação. Isto é tipicamente implementado em cima do UDP, mas outros protocolos básicos de comunicações podem ser igualmente usados.

O RTP define inicialmente a sintaxe da transmissão da informação, são usados perfis para descrever as semânticas, por exemplo como interpretar os campos dos cabeçalhos dos pacotes. Como resultado, os serviços RTP não estão completamente especificados, possibilitando que cada terminal de TV possa ajustar o protocolo para cumprir os seus requisitos. É também descrito um protocolo de controlo, o RTCP (*Real Time Control Protocol*), este é usado pelas aplicações para monitorizar a qualidade de serviço que a rede está a conseguir entregar, bem como para divulgar a informação sobre a sessão de comunicação que pertence aos intervenientes. De acordo com o foi anteriormente descrito, o RTP é um protocolo de com muita utilidade reconhecida para os terminais de TV que processam a transmissão de informação multimédia num domínio específico e independentemente do formato, podendo monitorizar e agir às alteração das condições da rede.

3.4.1 RTP/RTCP

O protocolo RTP existe como suporte às aplicações de tempo-real, não fazendo nenhuma reserva de recursos, caso seja necessário, ficando a cargo das camadas protocolares inferiores.

As principais especificações do protocolo RTP são:

- Proporciona mecanismos de transporte ponto-a-ponto apropriados para aplicações que operem na transmissão de dados em tempo-real;
- Independente do protocolo de transporte usado (caso este seja implementado sobre outro protocolo de transporte tradicional) e das camadas de rede que o suportam. Na prática deve ser usado um protocolo de transporte que implique um mínimo de controlo adicional e de largura de banda, como é o caso do UDP;
- No que diz respeito à camada de rede, poder-se-á dizer que está vocacionado para cenários de operação sem a necessidade de reserva de recursos por parte dos elementos da rede. Contudo, se esse serviço for fornecido, por exemplo pelo protocolo de reserva RSVP, o RTP continuará a desempenhar o papel de protocolo de transporte embora sem a vertente adaptativa;
- Possibilita mecanismos de comunicação em ambientes *multicast* ou *unicast*;
- Engloba um protocolo de controlo adicional (RTCP), que permite controlar o estado das ligações.

Vamos apresentar um conjunto de informações de estado de cada uma das entidades participantes. Na especificação da utilização do protocolo RTP ficou definido que na transmissão simultânea de vários fluxos de média, cada um deles deveria ser transmitido num canal RTP independente. O RTP foi concebido com o intuito de operar sobre mecanismos de *multicast* de uma forma a reduzir o tráfego da sessão, em particular no que se refere à informação de controlo RTCP.

I. Formato de um pacote RTP

Um pacote RTP está dividido num cabeçalho e na parte dos dados. No cabeçalho encontra-se a informação sobre o tipo de dados transportados (*payload type*) bem como informação útil para funcionalidades próprias do mecanismos do RTP. Na Figura 17 é apresentado o formato de um pacote RTP [21].

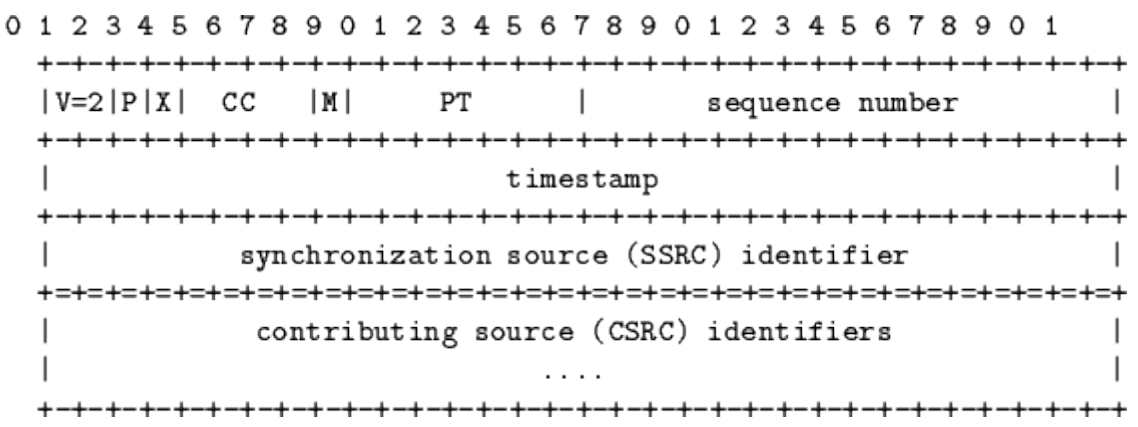


Figura 17 - Formato de um pacote RTP.

-Version (V) : Este campo define a versão do protocolo RTP que está a ser usado.
-Padding (P): Bit que quando ativado indica que o pacote contém informações adicionais que não fazem parte da codificação transmitida.
-Extension (X): Bit que quando ativado indica que existe um cabeçalho adicional para ser interpretado mediante o tipo de conteúdo transportado no pacote (identificado pelo *payload type*).
-CSRC count (CC): Indica o número de fontes de contribuição que operaram sobre

o canal RTP. Estes indicadores são usados no caso da participação dos denominados mixers, conversores de formato, que são sistemas intermediários que combinam e tratam os fluxos RTP originados por outros sistemas.

-Marker (M) : Interpretado consoante o perfil de média transportado e a sua codificação. Por exemplo, no caso de transporte de vídeo, indicar o fim de uma imagem que faz parte da sequência transmitida.

-Payload Type (PT): Identificador único que terá correspondência para um dado *payload format*, que definirá como a aplicação deverá tratar a informação recebida. No RTP a um dado *payload type* está associado um *payload format* que identifica como um determinado o perfil de média se encontra encapsulado no pacote RTP e como deve ser interpretado pela aplicação.

-Sequence Number: Identificador do número de pacote RTP. Sempre que for enviado um pacote RTP, este campo é incrementado de uma unidade pelo emissor.

-Timestamp: Estampilha temporal que indica o instante a que a primeira amostra do *payload* foi gerada, a frequência do relógio depende do *CoDec* usado (normalmente 8Khz), deve ser calculada de uma forma monótona e linear de modo a que seja possível ao recetor efetuar a sincronização do perfil de média em questão e o cálculo do *jitter* que está a ocorrer.

-SSRC: Identifica a entidade responsável pelo número de sequência e pelo *timestamp*, normalmente o emissor (número aleatório). O identificador de sincronização deverá

possibilitar ao recetor de uma sessão RTP agrupar as tramas de um dado emissor para posterior processamento.

-CSRC: Contem o SSRC de uma fonte da sessão. Usado quando a origem é um misturador.

II. Multiplexagem de sessões RTP

O protocolo RTP utiliza como método de multiplexagem do endereço de transporte, ou seja, um endereço de rede mais uma porta de destino definem uma única sessão RTP. Nessa sessão irão transferir-se pacotes RTP de um único tipo (definido pelo *payload type*), e originados por uma ou mais entidades (identificadas pelo SSRC). Caso as entidades troquem entre si outros tipos de dados, uma nova sessão RTP, independente da primeira, deverá ser utilizada.

III. RTCP - Real Time Control Protocol

Um dos atributos do RTP é a definição de um protocolo de controlo a ele associado. O RTCP é identificado pelo mesmo endereço de sessão RTP e pela porta de transporte da sessão RTP+1. No canal RTCP serão trocadas mensagens de controlo entre os diversos intervenientes na sessão RTP associada. Essas mensagens poderão ser de identificação das fontes (como o nome dos utilizadores participantes, endereços *e-mail*, etc.) ou então mensagens do estado da sessão. Estas últimas são geradas por todos os participantes da sessão e tem como principal objetivo descrever os valores dos diversos parâmetros de funcionamento (perdas de pacotes, variação nos atrasos, estado de sincronização, etc.) verificados por cada uma das entidades.

Existem dois tipos de relatórios que são gerados pelas entidades: SR (*Sender Reports*): gerados pelas entidades que são simultaneamente recetores e emissores RTP; e RR (*Receiver Reports*): gerados por entidades que são unicamente recetores RTP. Em ambos os casos, esses relatórios incluem informação de estado referente a todos os emissores RTP das quais as mensagens estão a ser processadas pela entidade que gera esses mesmos relatórios. Este é o mecanismo que permite ao emissor ter conhecimento do decorrer da aplicação na perspetiva dos recetores de informação, permitindo a estes obterem informações úteis para efetuarem alguns ajustes ao seu funcionamento.

IV. Formato de um pacote RTCP

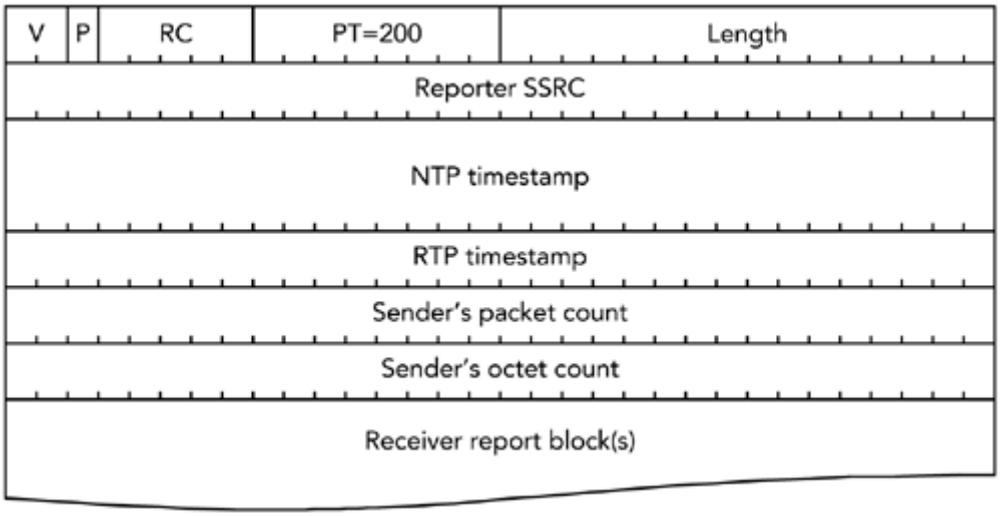


Figura 18 - Pacote RTCP.

Na Figura 18 estão apresentados os campos de um pacote RTCP e será feita uma breve descrição do seu significado, começando pelo cabeçalho.

- Version: Versão do protocolo.
- Padding: Quando ativado indica se este pacote contém octetos adicionais situados no fim do pacote RTCP.
- RC (*Reception Report Count*): Número de blocos presentes neste pacote que são estatísticas de receção para fontes de informação RTP.
- Length: Tamanho do pacote RTCP em número de palavras de 32 bits menos um.
- SSRC: Identificador da entidade que gerou este relatório.

V. Informação da própria entidade emissora

Estas informações estão relacionadas com a própria entidade que gerou este relatório.

- NTP Timestamp: Estampilha temporal global do sistema. A sua utilização é útil para efetuar a sincronização de diferentes sessões RTP.
- RTP Timestamp: Estampilha temporal enviada no último pacote RTP processado.
- Sender's Packet Count: Número total de pacotes enviados pelo emissor até ao momento que gera este relatório.
- Sender's Octet Count: Número total de octetos enviados até ao momento em que gera este relatório.

VI. Estatísticas das fontes

Os campos que a seguir que se descrevem referem-se a uma determinada fonte emissora de tráfego RTP.

- SSRC n: Identificador da fonte à qual dizem respeito as estatísticas presentes neste bloco.
- fraction lost: A percentagem de pacotes, gerados pela fonte em questão, que foram perdidos desde da geração do último pacote SR ou RR.
- cumulative number of packet losts: Número total de perdas de pacotes verificadas desde o início da sessão.
- higest number received: Último número de sequência recebido em pacotes RTP gerados pela fonte em questão.
- Jitter: Uma estatística da variação de chegada de pacotes RTP gerados pela fonte (nas mesmas unidades das estampilhas RTP).
- Last SR: Estampilha temporal do último relatório gerado pela fonte.
- DLSR: Diferença entre o instante em que recebeu o último SR (*Sender Report*) por parte da fonte, e o instante em que se está a gerar este pacote.

CAPÍTULO 4 – Trabalho desenvolvido

No capítulo anterior descrevemos os protocolos de descrição de serviços, protocolos de descrição de sessão bem como de transporte e de descrição das características dos terminais. Neste capítulo, vamos descrever todo o trabalho desenvolvido e implementado no Terminal de TV do projeto SUIT.

A inovação nas redes de telecomunicações e a mobilidade oferecida pelos operadores, tornam inevitáveis novos terminais com novas funcionalidades. Em particular, poder assistir em direto aos canais de TV oferecidos pelo operador em movimento, em múltiplos dispositivos diferentes com possibilidade de aceder a um *website* ou fazer uma chamada VoIP.

Com a massificação das tecnologias multimédia, não faz sentido um terminal de TV apenas reproduzir uma emissão, seja ela em direto ou em diferido. Nos dias de hoje, um terminal de TV tem que ter a capacidade da mobilidade, múltiplas funcionalidades muito para além das emissões de TV em direto, como por exemplo, ter *internet* e VoIP.

Para que isto seja possível, os terminais de TV tendem a ser uns autênticos mini computadores a correr um sistema operativo (*Linux* ou *Windows*) com um processador gráfico dedicado (GPU). Esta evolução deve-se ao facto de ser possível produzir processadores cada vez mais eficientes, com vários núcleos, e mais pequenos, bem como GPU's com capacidades de descodificação de pelo menos 25 fps.

Neste contexto, o trabalho desenvolvido consistiu no desenvolvimento de uma aplicação que corre no sistema operativo *Windows*, mas com a possibilidade de portabilidade para outros sistemas operativos (por exemplo *Linux*) capaz de descodificar o vídeo escalável enviado pelo *playout* em tempo real bem como fazer um pedido de um vídeo alojado no *playout*, um *browser* para visualizar páginas web e a incorporação do *Skype* para chamadas VoIP.

Posto isto, neste capítulo 4, vamos descrever todo o processo que foi executado na elaboração do terminal de TV e os resultados que se obtiveram em diversos testes feitos, quer em laboratório, quer no exterior e em movimento.

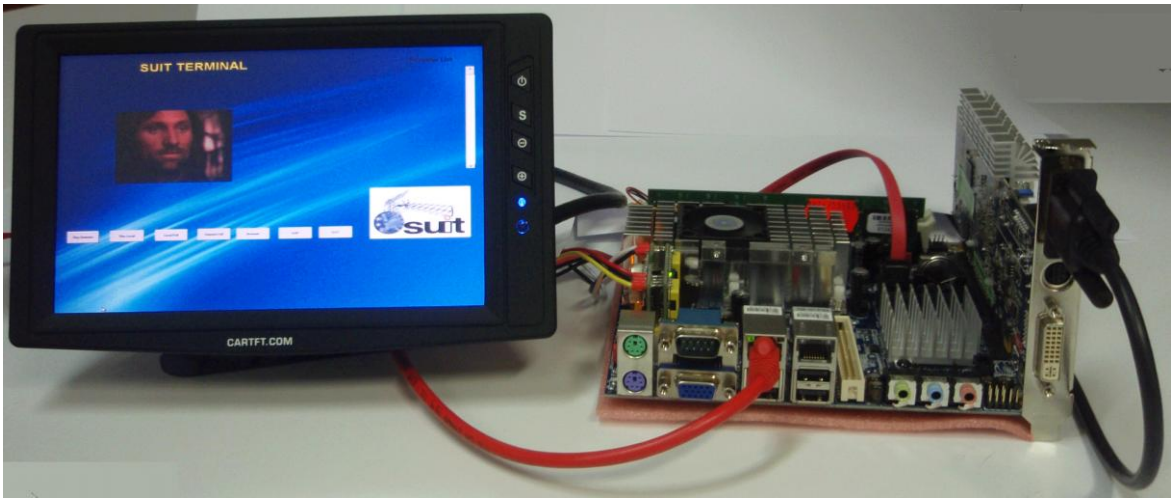


Figura 19 - Terminal de TV.

Na Figura 19 podemos observar o *hardware* do terminal de TV. Do lado esquerdo encontra-se um ecrã *Full* HD tátil onde é apresentada a parte gráfica ao utilizador. Nesta figura, o vídeo está a reproduzir-se do disco local. Do lado direito da figura, está uma *board* mini-atx com uma placa gráfica com capacidade de descodificação até resoluções *Full* HD.

4.1 Funcionalidades implementadas

Em síntese, as funcionalidades implementadas fazem deste terminal de TV um autêntico sistema multimédia.

Desde logo porque, graças ao canal de retorno do DVB-RCT e WiMAX é possível oferecer serviços como internet e o VoIP.

O terminal de TV está assim preparado para fornecer os seguintes serviços:

- Ver emissões em tempo real.

- Aceder a uma lista de vídeos armazenados no *playout.*

- Adaptado a vários dispositivos.

- Possibilidade de pause.

- Acesso à internet através do *browser*.

- Efetuar chamadas VoIP usando o *Skype*

Apenas foi implementado áudio nos serviços VoD. Foi escolhido a codificação *Advanced Audio Coding* (AAC). A sincronização do áudio e o vídeo será feita com os pacotes RTCP. Esta implementação será feita com o *software open source Freeware Advanced Audio Decoder* (FAAD2), estando escrito em linguagem C, estando disponível para a plataforma *Windows* e *Linux*.

4.2 Interface da aplicação

A interface do terminal de TV foi desenvolvida na plataforma de desenvolvimento pertencente à *Microsoft* e chama-se *Microsoft Visual Studio*. A linguagem de programação utilizada foi o C/C++. Será aqui que o utilizador escolhe o serviço que quer ver de acordo com os serviços disponibilizados pelo operador. Parte do código-fonte pode ser visto no anexo A.3 Código-fonte principal da aplicação.

Figura 20 - Ambiente gráfico do terminal de TV.

A Figura 20 mostra o terminal do utilizador final em execução. Na Figura 20, é possível ver no lado direito os serviços oferecidos por um operador, neste caso, o operador de serviços possui apenas alguns serviços, mas pode ter muitos mais. O terminal de TV apresenta o serviço pedido no lado esquerdo, sendo este amostrado de acordo com o pedido feito ao *playout* tendo em conta o tipo dispositivo que é, terminal móvel, SDTV ou HDTV.

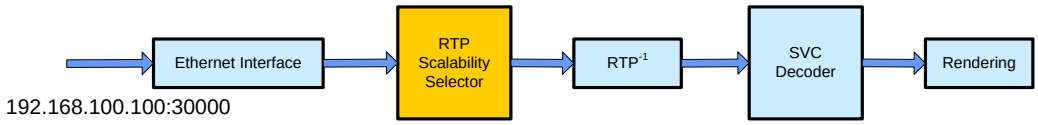


Figura 21 - Arquitetura do terminal de TV.

Estas são as etapas pelas quais o terminal de TV passa até que seja visualizado o serviço pedido:

- Recebe o fluxo RTP via rede Wi-Fi.

- O RTP é desencapsulado.

- Decodificado usando o codec SVC.

- Finalmente a visualização.

O terminal de TV é capaz de reproduzir vídeos locais, isso será usado quando o utilizador final deseja gravar a emissão para vê-la mais tarde. Iremos detalhar estas etapas mais em detalhe no ponto seguinte.

Na aplicação existe um botão que abre um *browser* desenvolvido para o projeto e um botão que chama a aplicação *Skype*.

4.3 Protocolos implementados

De modo a que o terminal de TV pudesse comunicar com o *playout* foi necessário a utilização de vários protocolos, entre eles: RTP/RTCP, RTSP, IGMP, SD&S, MPEG21-DIA e SDP. Estes protocolos foram descritos no capítulo 3.

Os diagramas de sequência abaixo mostram as etapas que o terminal de TV executa quando é inicializado, consoante se trate de uma ligação *unicast* ou *multicast*.

Primeiro, o terminal encontra o serviço SD&S, executando a descoberta do serviço SD&S para localizar os serviços do operador.

O modo como se procede à localização do servidor SD&S varia, consoante seja *multicast*, Figura 22 ou *unicast* Figura 23.

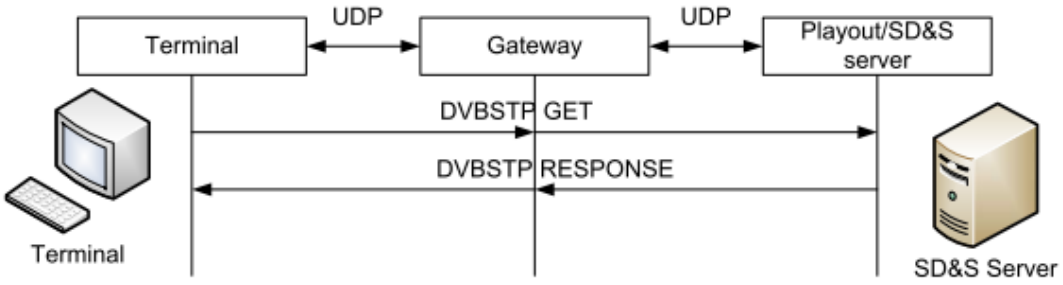


Figura 22 - Localização do servidor SD&S *multicast*.

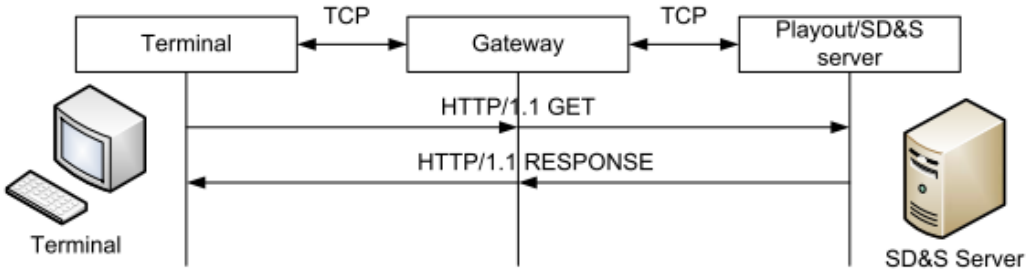


Figura 23 - Localização do servidor SD&S *unicast*.

Depois o terminal receber a lista de serviços do operador disponíveis no *playout*, o utilizador final escolherá o programa ou serviço que gostaria de assistir no terminal de TV clicando na lista obtida do operador que aparece no lado direito do terminal de TV que contém os serviços. Apenas foi implementado o modo de localização do servidor SD&S *unicast*. Os serviços *multicast* encontram-se descritos num ficheiro terminal.ini com os endereços *multicast* dos respetivos serviços.

Dependendo do serviço oferecido pelo operador, o terminal pode juntar-se a uma sessão *multicast* (IGMP) mostrada na Figura 24 ou solicitar uma sessão *unicast* (RTSP) como se mostrada na Figura 27.

No cenário de VoD o serviço enviado pelo *playout* não é escalável, sendo por isso, codificado em H.264/AVC.

Pode observar-se na Figura 24 o envio de duas descrições SVC, sendo a D1 composta pelas camadas CIF,SD e HD e a D2 apenas a camada base CIF.

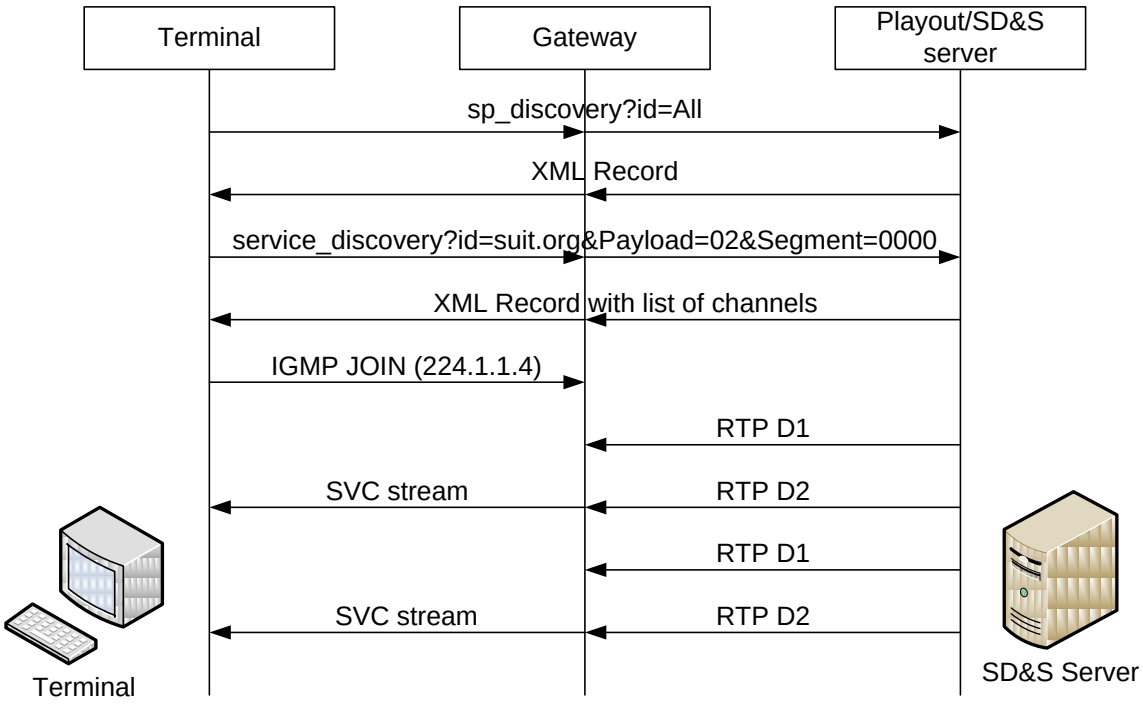


Figura 24 - Diagrama de sequência de arranque do Terminal de TV num cenário *multicast*.

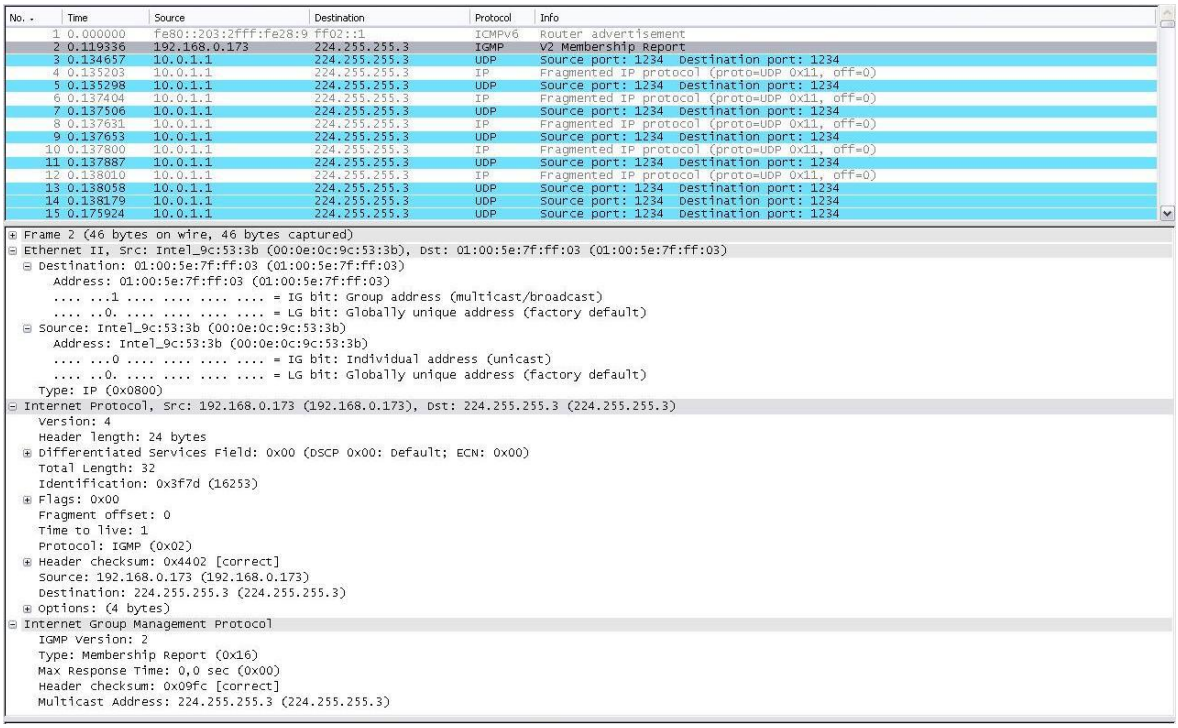


Figura 25 - Pedido para se juntar a uma sessão *multicast*.

Na Figura 25, podemos ver a primeira troca de pacotes que corresponde ao pedido IGMP do terminal de TV para se juntar ao grupo *multicast*.

Depois do terminal de TV se juntar ao grupo *multicast*, começa a receber pacotes UDP do *playout*, ver Figura 26. Estes pacotes têm o IP de origem do *playout* e como destino o endereço de IP *multicast* 224.255.255.3. Ao nível do endereço *media access control* (MAC), os pacotes *user datagram protocol* (UDP) têm origem no MAC do *access point* (AP).

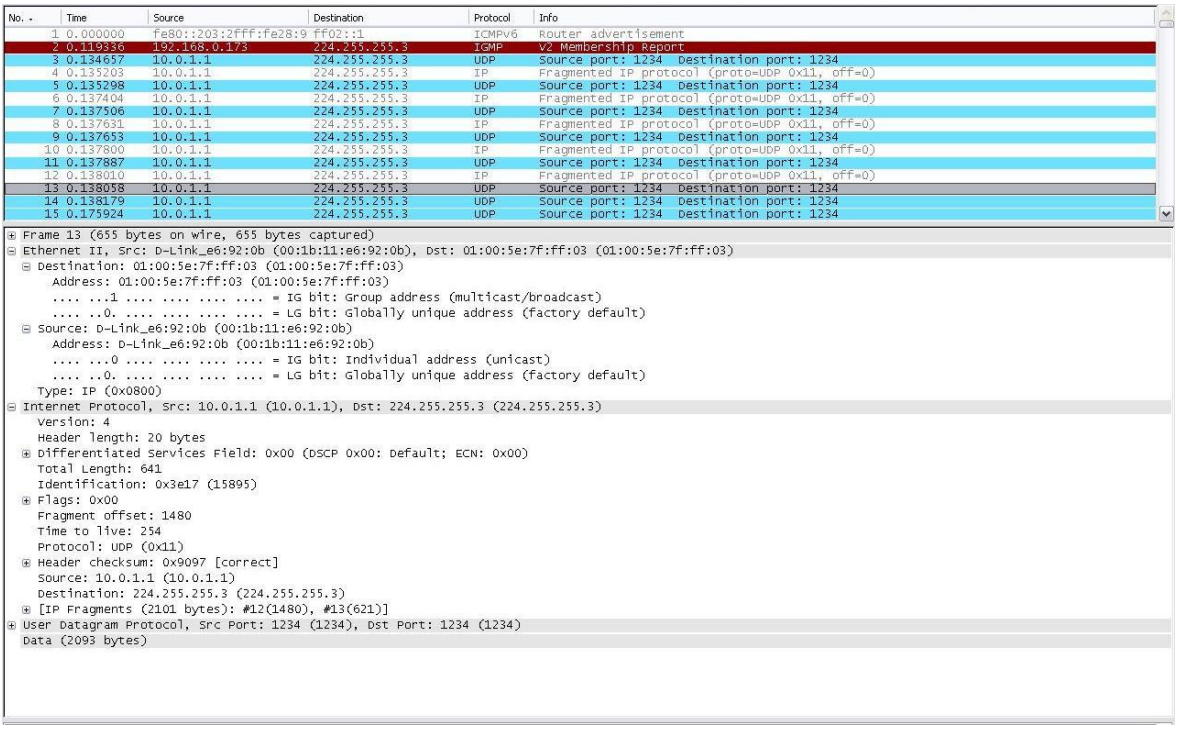


Figura 26 - Pacote UDP.

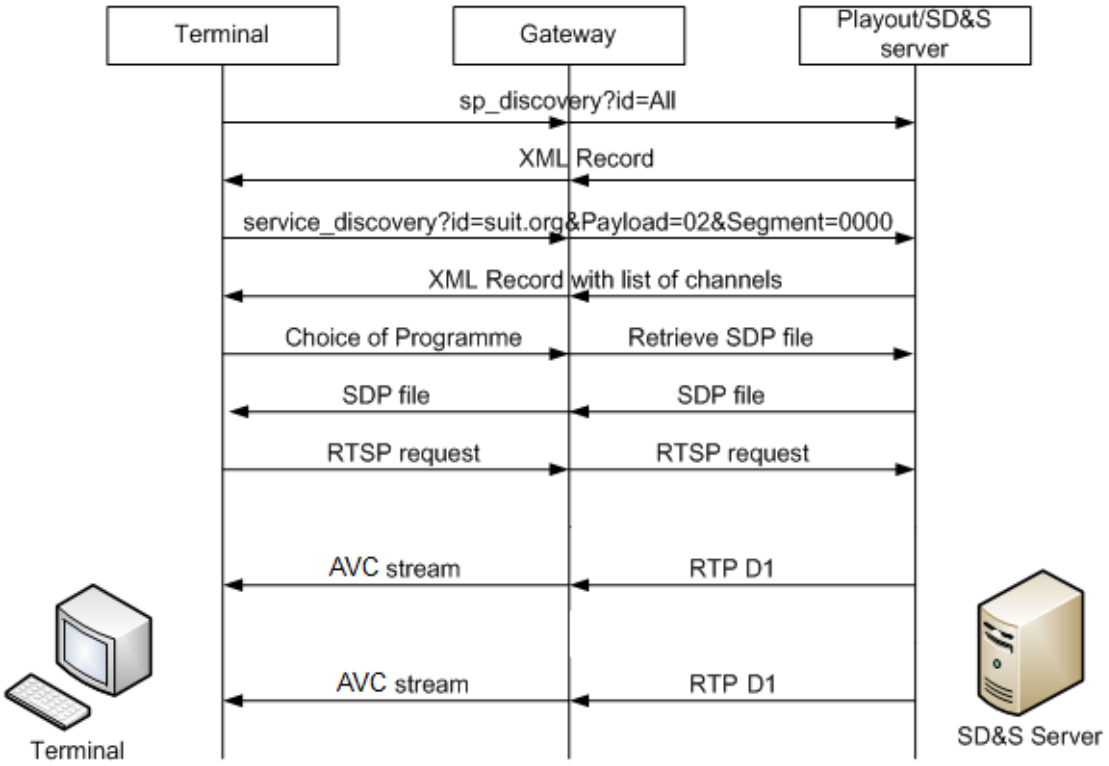


Figura 27 - Diagrama de sequência de arranque do Terminal de TV num cenário *unicast*.

No caso do cenário *unicast* apresentado na Figura 27, o serviço que é oferecido ao utilizador final é o VoD.

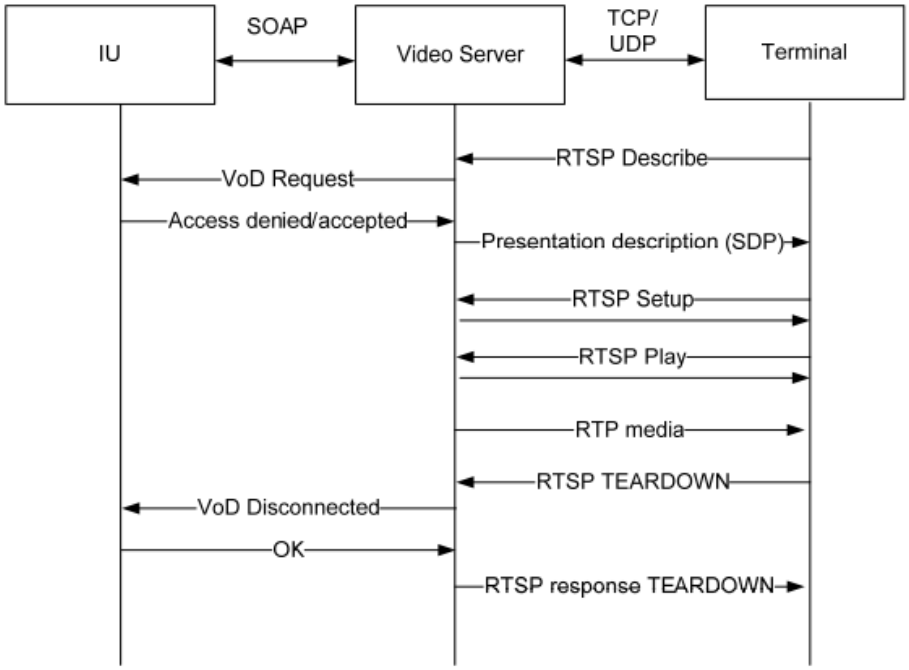


Figura 28 - Diagrama do serviço VoD.

À parte da largura de banda disponível, o terminal terá que informar o *playout* sobre outras características necessárias. As mais importantes, resolução do ecrã e espaço em disco. Esta informação é transmitida ao *playout* usando o protocolo MPEG-21 DIA UED, enviando um código XML ao *playout* com essa informação. No disco do terminal encontra-se um ficheiro rcic.ini, ver anexo A.2 Ficheiro rcic.ini, que contém as informações do terminal de TV, entre outras, a resolução do terminal que é lida para posteriormente ser criado o código XML e enviado ao *playout*. O espaço em disco é calculado por uma função específica no terminal.

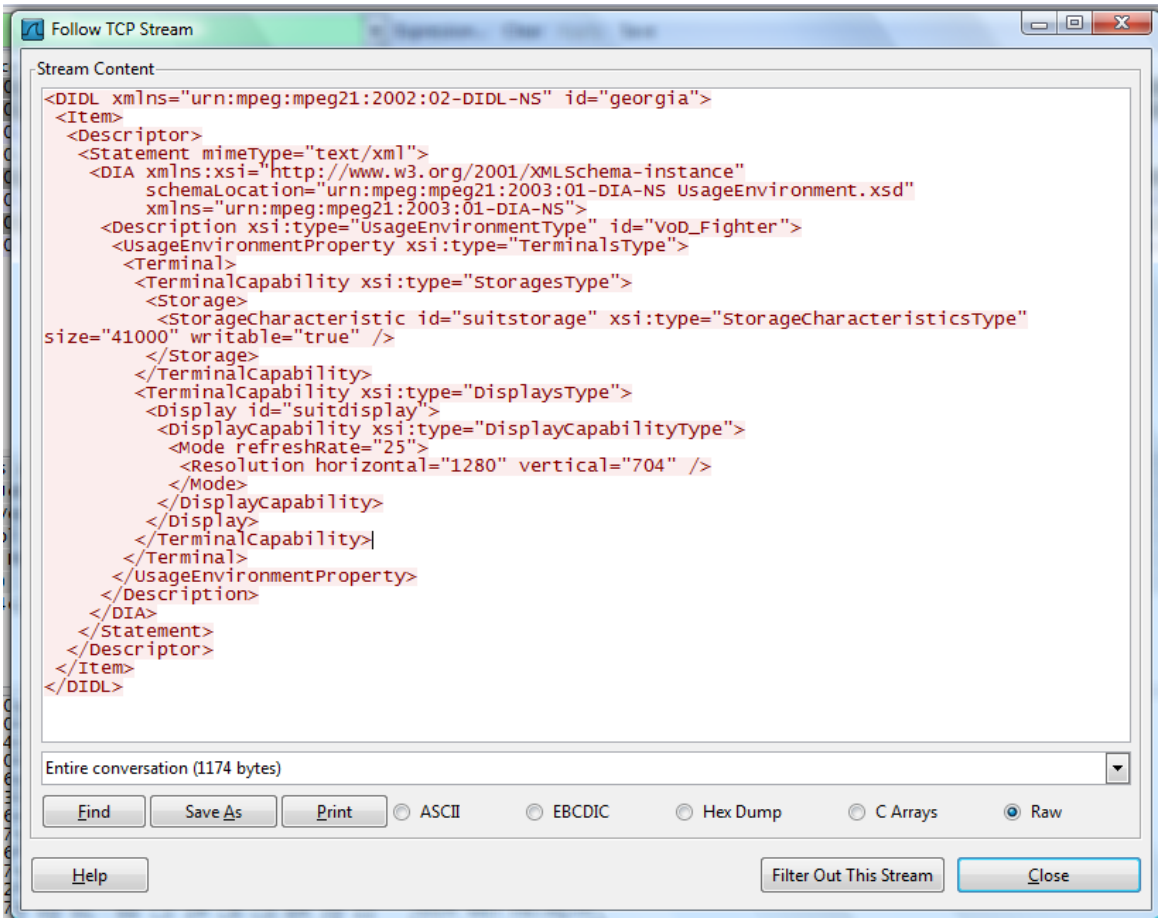


Figura 29 - Código XML MPEG-21 DIA UED enviado pelo terminal.

Na Figura 29 vemos o terminal a enviar as suas características como o espaço em disco bem como a resolução para o *playout*. Desta forma o *playout* pode otimizar a utilização da largura de banda disponível.

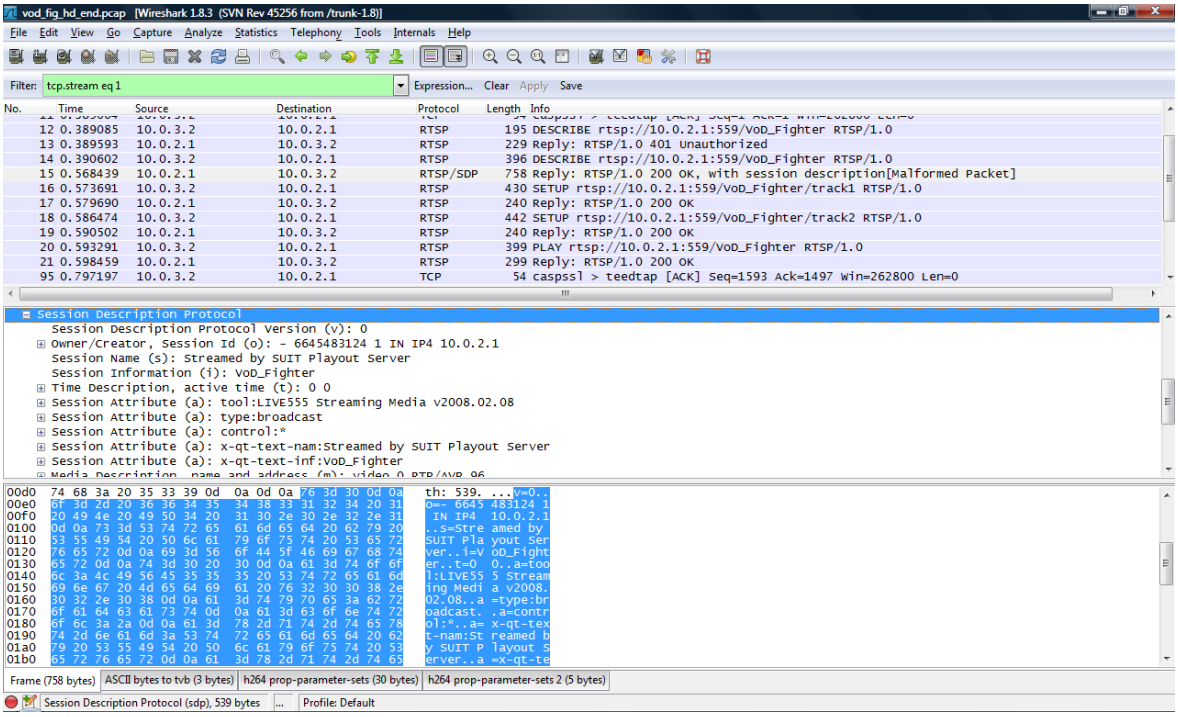


Figura 30 - Captura da inicialização de um serviço VoD.

Na Figura 30 podemos ver a inicialização de um serviço VoD pelo terminal de TV. Como se pode ver, o protocolo utilizado é o RTSP. Em primeiro lugar o terminal de TV faz um pedido *DESCRIBE*, como se pode ver a seguir, de forma a saber quais são os recursos disponíveis no *playout*.

```
DESCRIBE rtsp://10.0.2.1:559/VoD_Fighter RTSP/1.0

Accept: application/sdp

Authorization: Digest username="georgia", realm="LIVE555 Streaming Media",
nonce="7a9919d2d0ae304ee5f043ecb709f87a",
URL="rtsp://10.0.2.1:559/VoD_Fighter",
response="812b8be4cfe58a95b3a219020b418dca"

User-Agent: terminal.exe (SUIT Terminal v2008.05.12)
```

O *playout* envia através do envio de um SDP essa informação, como se mostra a seguir, para o terminal de TV.

```
v=0
o=- 6645483124 1 IN IP4 10.0.2.1
s=Streamed by SUIT Playout Server
i=VoD_Fighter
t=0 0
```

```
a=tool:LIVE555 Streaming Media v2008.02.08
a=type:broadcast
a=control:*
a=x-qt-text-nam:Streamed by SUIT Playout Server
a=x-qt-text-inf:VoD_Fighter
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
a=rtpmap:96 H264/90000
a=fmtp:96 packetization-mode=1;profile-level-id=4D4020;sprop-parameter-
sets=J01AIJZWCgz8+AiAABOIAAPQkHNgAnqAJ6zOfAQA,KM48gAA=
a=control:track1
m=audio 0 RTP/AVP 14
c=IN IP4 0.0.0.0
a=range:npt=0-145.604
a=control:track2
```

Após o terminal de TV receber o SDP, o terminal de TV envia ao *playout* um pedido *SETUP* que inicia uma sessão, ou seja, um canal de comunicação entre o terminal de TV e o *playout* bem como os atributos da sessão, enviando um pedido de SETUP por cada atributo, neste caso um para o vídeo e outro para o áudio.

```
SETUP rtsp://10.0.2.1:559/VoD_Fighter/track1 RTSP/1.0

Transport: RTP/AVP/UDP;unicast;client_port=5000-5001

Authorization: Digest username="georgia", realm="LIVE555 Streaming Media",
nonce="7a9919d2d0ae304ee5f043ecb709f87a",
URL="rtsp://10.0.2.1:559/VoD_Fighter/",
response="b26dc68fc83d574454e368a24a537d1e"

User-Agent: terminal.exe (SUIT Terminal v2008.05.12)
----------------------------------------------------------------------------------------------------
SETUP rtsp://10.0.2.1:559/VoD_Fighter/track2 RTSP/1.0

Transport: RTP/AVP/UDP;unicast;client_port=5002-5003

Authorization: Digest username="georgia", realm="LIVE555 Streaming Media",
nonce="7a9919d2d0ae304ee5f043ecb709f87a",
URL="rtsp://10.0.2.1:559/VoD_Fighter/",
response="b26dc68fc83d574454e368a24a537d1e"

User-Agent: terminal.exe (SUIT Terminal v2008.05.12)
```

Seguidamente o terminal de TV envia ao *playout* o pedido *PLAY*, desta forma o *playout* dá início ao envio do conteúdo multimédia, através de pacotes RTP, que lhe foi pedido.

```
PLAY rtsp://10.0.2.1:559/VoD_Fighter RTSP/1.0

Range: npt=0.000-

Authorization: Digest username="georgia", realm="LIVE555 Streaming Media",
nonce="7a9919d2d0ae304ee5f043ecb709f87a",
URL="rtsp://10.0.2.1:559/VoD_Fighter/",
response="6f9b7690193b14bf53a0ce783e214d4a"

User-Agent: terminal.exe (SUIT Terminal v2008.05.12)
```

Finalmente para terminar a sessão, o terminal de TV envia ao *playout* o pedido *TEARDOWN*.

```
TEARDOWN rtsp://10.0.2.1:559/VoD_Fighter RTSP/1.0

Authorization: Digest username="georgia", realm="LIVE555 Streaming Media",
nonce="7a9919d2d0ae304ee5f043ecb709f87a",
URL="rtsp://10.0.2.1:559/VoD_Fighter/",
response="5bd6e14ae935003b0493307dd1db1c70"

User-Agent: terminal.exe (SUIT Terminal v2008.05.12)
```

O utilizador final também tem a opção de fazer uma pausa no serviço VoD que pediu. Para isso o terminal envia ao *playout* o pedido *PAUSE*. Caso queria voltar a visualizar, o terminal de TV enviará um pedido *PLAY*. Caso contrário, encerrará a sessão, com um pedido *TEARDOWN*.

```
PAUSE  rtsp://10.0.2.1:559/VoD_Fighter RTSP/1.0

Authorization: Digest username="georgia", realm="LIVE555 Streaming Media",
nonce="7a9919d2d0ae304ee5f043ecb709f87a",
URL="rtsp://10.0.2.1:559/VoD_Fighter/",
response="5bd6e14ae935003b0493307dd1db1c70"

User-Agent: terminal.exe (SUIT Terminal v2008.05.12)
```

Na Figura 31 podemos ver um pacote RTCP enviado do terminal de TV para o *playout* capturado usando o *software* Wireshark. Como foi descrito anteriormente, o protocolo RTCP envia dados estatísticos que podem ser usados, por exemplo, para alterar a taxa de transmissão. Este pacote RTCP contém informação *Receiver Report* (RR) e *Source Description* (SDES) como se pode ver na respetiva figura.

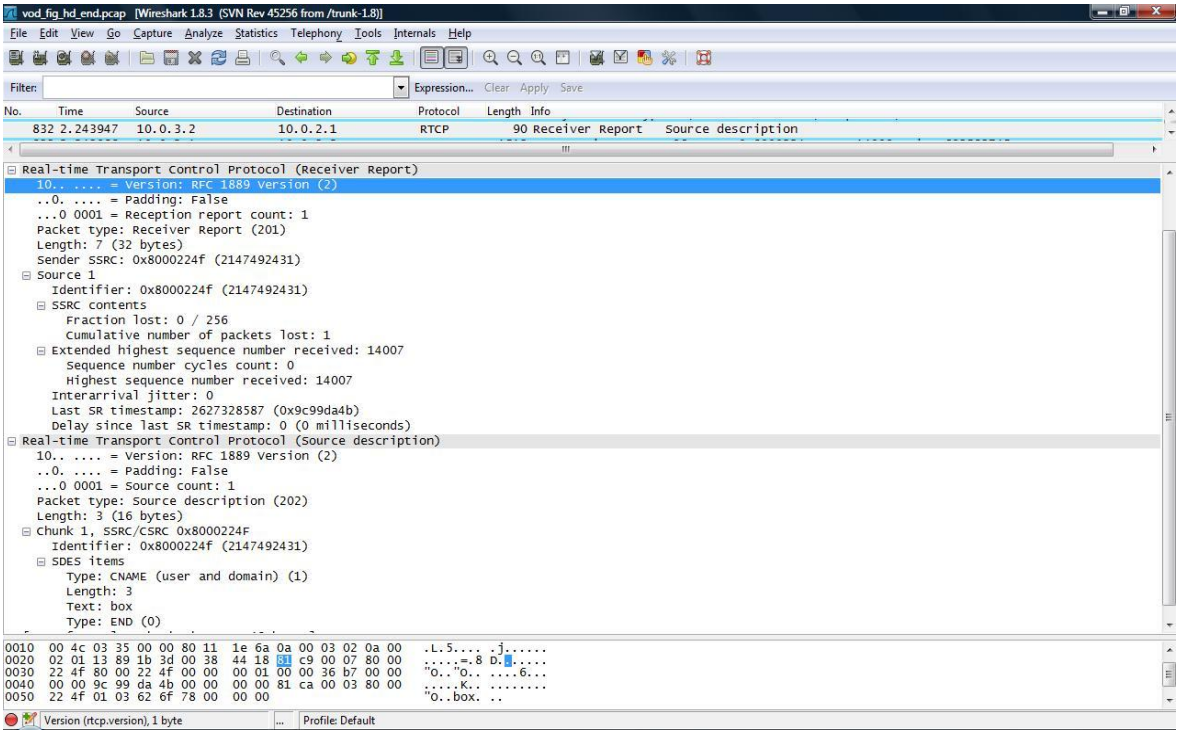


Figura 31 - Captura de um pacote RTCP

4.4 Desencapsulador IP

O desencapsulador encontra-se integrado no *software* do terminal de TV, sendo a sua principal função a extração da informação contida nos pacotes RTP enviados durante um emissão em tempo real ou do serviço VoD. Para entendermos a função do desencapsulador temos que compreender de que forma o playout envia o fluxo de multimédia. Aqui vamos distinguir duas situações, *multicast* e *unicast*. No caso do *multicast* o vídeo enviado é escalável e contém todas as camadas, CIF (base), SD e HD. Por sua vez, no caso do *unicast* é apenas enviado a camada pedida pelo terminal de TV, ou seja, o *playout* tem um extrator que extrai a camada que lhe foi pedida pelo terminal de TV, usando o MPEG-21 DIA UED como já foi descrito.

Os pacotes RTP recebidos são então, desencapsulados num *buffer* circular e descodificados como pacotes RTP e é extraída a informação do cabeçalho (numero de sequencia, *timestamp*) e um ponteiro para o conteúdo das *network abstraction layer unit* (NALUs) contidas nesse mesmo pacote.

As NALUs são processadas em bruto e é criado o objeto AnnotatedNALUnit, que vai incluir o tamanho da imagem, número da NAL ou o número da *slice* (caso a imagem esteja dividida em *slices*).

Para compreendermos como são desencapsuladas as NALUs, em algum detalhe, é necessário compreender como são encapsuladas [22].

Existem três modos de empacotamento das NALUs :

1. Um único pacote NAL
2. Agregação de pacotes (STAP-A)
3. Fragmentação de pacotes (FU-A)

Por norma, no formato HD e SD, as imagens I são fragmentadas e as imagens P são agregadas ou enviadas num único pacote NAL devido ao *maximum transmission unit* (MTU) máximo da rede. Contudo, por exemplo, num jogo de futebol onde existe grande movimento as imagens P poderão sofrer fragmentação.

Vamos explicar os modos de empacotamento das NALUs com algum detalhe.

1. Um único pacote NAL

Neste modo, cada NAL pertence a uma imagem encapsulada num pacote RTP, mantendo o mesmo *timestamp* em todos os pacotes RTP como se pode ver na Figura 32.

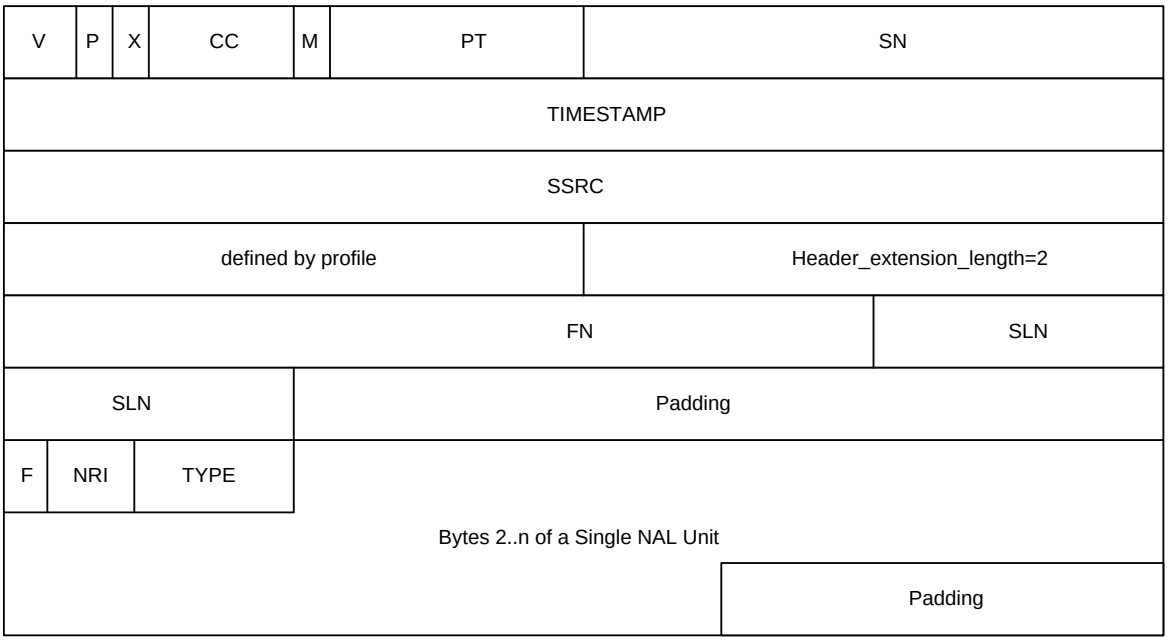


Figura 32 - Único pacote NAL.

V = 0 (versão)
P = 0
X = está ativo se existem parametros adicionais na extensão do cabeçalho RTP
CC = 0
M = está ativo se é a última NAL a ser enviada.

PT = 98 , se conteúdo do *payload* for H264/AVC e H264/SVC
SN = é incrementado de 1 por cada pacote RTP
TIMESTAMP = o mesmo que NAL *timestamp*.
SSRC = valor aleatório.
header_extension_lentgh = 2, indica a extensão de dois bytes no cabeçalho.
FN = número da imagem da NAL a que pertence.
SLN = número da *slice* que contém a NAL.
F = 0
NRI = 00 indica que o conteúdo da NAL não é usado para a reconstrução de imagens de referência.
TYPE = Tipo da NAL.

No caso da codificação SVC, é aplicada a extensão do cabeçalho e são adicionados dois bytes como mostra a Figura 33.

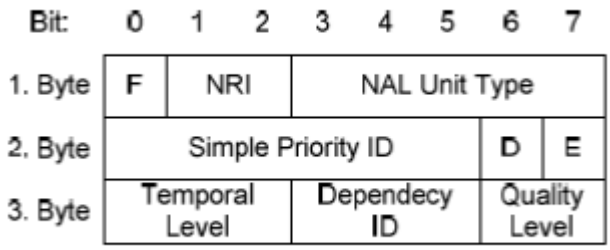
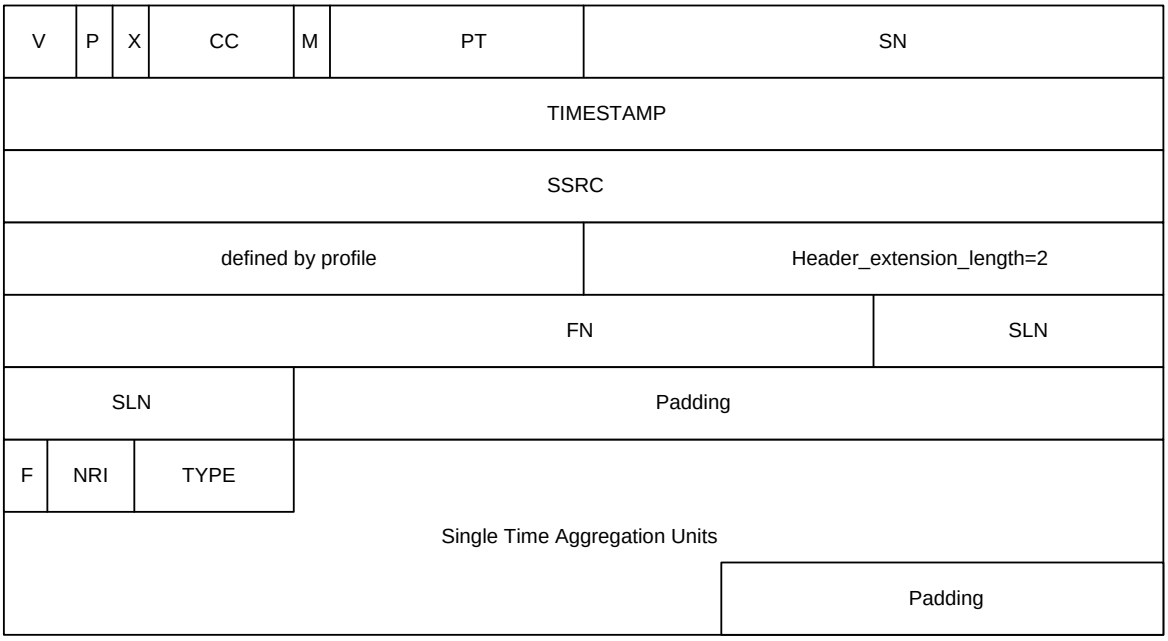


Figura 33 - Extensão do cabeçalho NAL SVC.

2. Agregação de pacotes (STAP-A)

No caso da agregação de pacotes, isto significa, que é possível agregar um ou mais NALUs sem exceder o MTU máximo da rede.



Com a agregação:

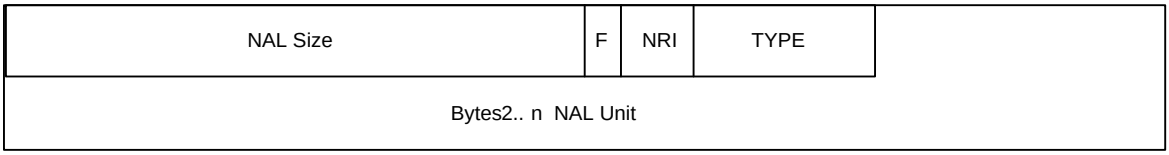


Figura 34 - Agregação de pacotes.

V = 0 (versão)
P = 0
X = está ativo se existem parametros adicionais na extensão do cabeçalho RTP
CC = 0
M = está ativo se é a última NAL a ser enviada.
PT = 98 , se conteúdo do *payload* for H264/AVC e H264/SVC
SN = é incrementado de 1 por cada pacote RTP
TIMESTAMP = o mesmo que NAL *timestamp*.
SSRC = valor aleatório.
header_extension_lentgh = 2, indica a extensão de dois bytes no cabeçalho.
FN = número da imagem da NAL a que pertence.
SLN = número da *slice* que contém a NAL.
F = 0
NRI = número máximo de NAL a serem transportadas.
TYPE = pacote STAP-A Tipo 24.

No caso da codificação SVC o cabeçalho adicionar é igual ao da Figura 33.

3. Fragmentação de pacotes (FU-A)

Caso seja detetada uma NAL superior ao MTU máximo da rede, esta é fragmentada como se pode ver na Figura 35.

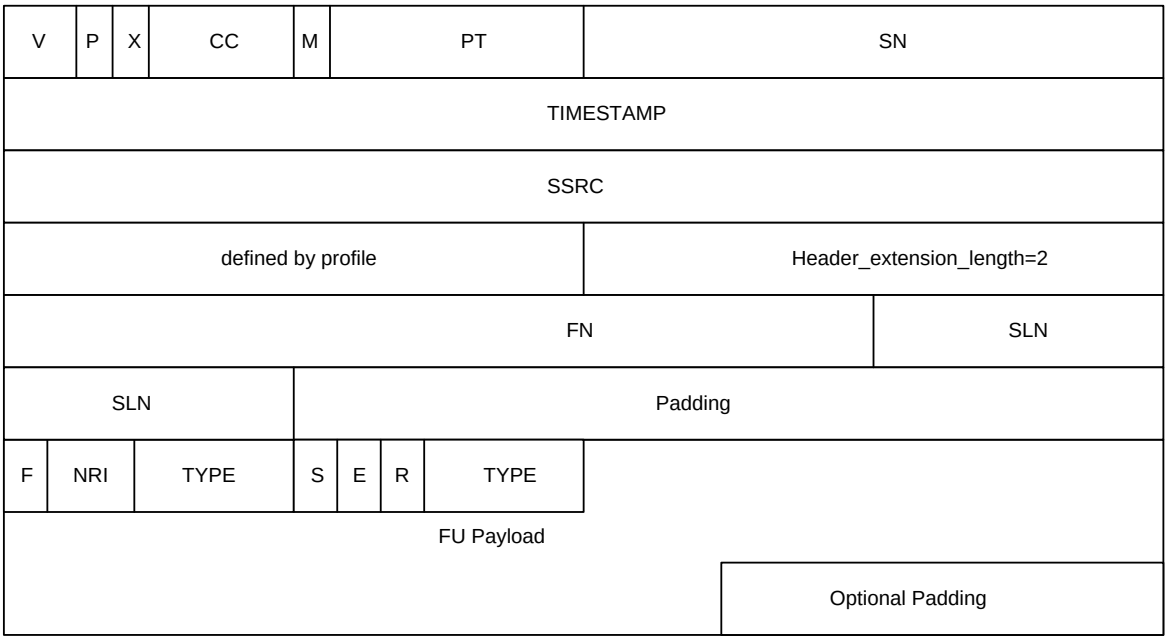


Figura 35 - Fragmentação de pacotes.

V = 0 (versão)
P = 0
X = está ativo se existem parametros adicionais na extensão do cabeçalho RTP
CC = 0
M = está ativo se é a última NAL a ser enviada.
PT = 98 , se conteúdo do *payload* for H264/AVC e H264/SVC
SN = é incrementado de 1 por cada pacote RTP
TIMESTAMP = o mesmo que NAL *timestamp*.
SSRC = valor aleatório.
header_extension_lentgh = 2, indica a extensão de dois bytes no cabeçalho.
FN = número da imagem da NAL a que pertence.
SLN = número da *slice* que contém a NAL.
F = 0
NRI = 00 indica que o conteúdo da NAL não é usado para a reconstrução de imagens de referência.
TYPE = pacote FU-A Tipo 28.

No caso da codificação SVC o cabeçalho adicionar é igual ao da Figura 33, mas apenas está presente no primeiro fragmento NAL enviado, ou seja, S=1.

Adicionalmente têm os seguintes parâmetros:

S = está ativo se está a ser enviado o primeiro fragmento da NAL
E = está ativo se está a ser enviado o último fragmento da NAL
R = 0

As imagens podem ser do tipo I, P ou B. No caso particular foram usadas apenas imagens I e P, uma pequena descrição dos tipos de imagens:

- Imagens I (intra pictures) – As imagens I são codificadas sem terem nenhuma ligação a outras, parecido com imagens JPEG. Ou seja, as imagens I possuem toda a informação necessária para a reconstrução pelo descodificador; por este motivo, as imagens I são o inicio de uma sequência de vídeo. A taxa de compressão das imagens I é muito reduzida e é idêntica à taxa de compressão de uma imagem JPEG com a mesma resolução.

- Imagens P (predicted pictures) – As imagens P são codificadas usando técnicas de predição com compensação de movimento, usando imagens de referências anteriormente codificadas, sendo estas do tipo I ou to tipo P.

Imagens P podem ser utilizadas como base para imagens P seguintes, mas como a compensação de movimento não é ideal não é possível aumentar muito o número de imagens P entre duas imagens I. A taxa de compressão de imagens P é elevada e superior à taxa de compressão das imagens I.

- Imagens B (bidirectional predicted pictures) – As imagens B são codificadas por interpolação bidireccional com inicio em imagens I e P anteriores e posteriores. Como as imagens B não são utilizadas para codificar imagens posteriores elas não propagam erros de codificação, contudo têm de estar armazenadas num buffer, implicando um atraso na reprodução de um video em tempo real. As imagens B têm a taxa de compressão maior.

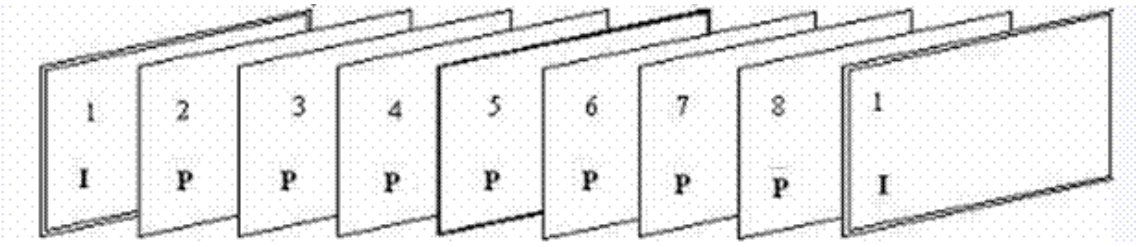


Figura 36 - Grupo de imagens.

Uma série de imagens de vídeo é subdividida em Grupos de Imagens (GOP's - *Groups of Pictures*),como se pode ver na Figura 36. Cada GOP é iniciado com uma imagem do tipo I, posteriormente pode conter varias imagens do tipo P e B. Assim sendo, é possível editar uma sequência de imagens tendo em contado o início do GOP. Não havendo erros de transmissão, podemos dizer que não ocorre propagação de erros. Nesse caso podemos prescindir das imagens do tipo **I**. Utilizamos apenas predição progressiva (Imagens **P**), baseada na imagem **P** anterior reconstruída. Apenas a primeira imagem da sequência será do tipo **I**.

A robustez do terminal de TV advém de, havendo erros no canal de transmissão, não os passar para o utilizador final. Para isso foi implementado um algoritmo robusto no processamento das imagens recebidas nos pacotes RTP. Será feita uma breve descrição do algoritmo.

Através do pacote RTP podemos extrair informação muito relevante como é o campo do número de sequência. Desta forma, sempre que é detetado um número de sequência que não é contínuo, essa imagem que estava a ser recebida é descartada sendo apresentada ao utilizador a imagem anterior que se encontrava no *buffer* até que seja recebida uma nova imagem do tipo I. Foram feitos testes de forma a avaliar a viabilidade da utilização das imagens P, contudo, a imagem mostrava muito arrastamento não sendo aceitável apresentar a imagem ao utilizador final.

Figura 37 - Imagem com perda de pacotes.

Como podemos ver pela imagem da Figura 37, num evento de perda de pacotes, existe a propagação de erros nas imagens P apresentado uma imagem com muitos erros em que é difícil identificar o que se está a ver. Além disso, o descodificador pode bloquear colocando o terminal de TV em sério risco de ter que ser necessário inicializa-lo.

Em suma o algoritmo de cancelamento de imagem é processado da seguinte forma:

1. É detectada um perda de um pacote RTP através do número de sequência.
2. A última imagem processada que se encontra no *buffer* é amostrada.
3. Através do *payload* do RTP, no caso particular, NALUs, aguardamos até recebermos a primeira imagem I ou fragemento da mesma, analisando o cabeçalho da imagem. Imagem I – 001 e imagem P – 010 , bits 3,4 e 5.

byte 4								byte 5								byte 6								byte 7							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
temperal sequence number								frame type 1=I, 2=P 3=B, 4=D								VBV delay								---							

Figura 38 - Cabeçalho da imagem

4. No caso de ser enviada sem fragmentação é imediatamente descodificada e amostrada, se tiver fragmentação aguardamos até receber todos os fragmentos para posteriormente ser descodificada e amostrada.

Figura 39 - Imagem sem perda de pacotes.

A Figura 39 apresenta a mesma imagem que a Figura 37, mas sem perda de pacotes. Ou seja, por exemplo, havendo perda de pacotes a seguir está imagem será amostrada ao utilizador final até que seja recebida uma nova imagem I completa.

Tendo em conta que o GOP são 25 imagens, numa situação extrema em que existem muita perda de pacotes ou deixa de haver receção de pacotes o utilizador final notará a imagem

parada, já numa situação com pouca perda de pacotes o utilizador final não notará nada, porque não será percetível ao olho humano.

As imagens serão depois descodificadas usando o *CoDec* SVC e amostradas ao utilizador final, será o assunto abordado a seguir. Partes do código-fonte podem ser vistas no anexo A.4 Código-fonte do desencapsulador.

4.5 Visualização

As imagens depois de desencapsuladas chegam à última fase da cadeia. Aqui as imagens são descodificadas e amostradas ao utilizador final, quer seja num ecrã de *toutch*, numa televisão ou mesmo num telemóvel.

O *CoDec* SVC tem vários parâmetros, porém, o foco será o parâmetro *spatial*. Este parâmetro diz ao descodificado SVC com que qualidade espacial deve descodificar o vídeo, por outras palavras, qual a camada a ser utilizada se CIF, SD ou HD. Por defeito a camada que é descodificada é a base, neste caso, CIF.

Este parâmetro tem importância no caso do *multicast*, porque como já foi aqui referido anteriormente, no caso do *unicast* o terminal de TV informa o *playout* quais as suas capacidades à priori, não sendo necessário indicar qual a camada para ser descodificada.

O *CoDec* SVC tem por base o *ffmpeg*, modificado para o SVC de forma a permitir escalabilidade no vídeo. O retorno que se obtém é uma imagem YUV que pode ser visualizada através da biblioteca SDL.

Para a amostragem das imagens foi utilizada a biblioteca SDL[3] (Simple DirectMedia Layer) em detrimento do DirectX pois o SDL é uma biblioteca livre e de código aberto, multiplataforma o que significa que pode ser utilizada em sistemas operativos *Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX,* e *QNX*.

O modo de utilização da biblioteca SDL é muito simples.

Deve-se incluir no cabeçalho da aplicação a seguinte declaração:

```
#include <SDL/SDL. h>
```

[3] O SDL pode ser obtido no endereço www.libsdl.org.

De seguida, inicializamos a biblioteca SDL. Isso é feito chamando a função:

```
SDL_Init();
```

Antes de passarmos ao próximo passo, uma breve explicação sobre "Surfaces" do SDL. Uma "*surface*" é uma área de memória utilizada para armazenar imagens.

Para declarar uma *surface*, usamos:

```
SDL_Surface *screen;
```

Para inicializar a *surface* como ecrã principal usamos:

```
screen = SDL_SetVideoMode(640,480,32,SDL_HWSURFACE|SDL_DOUBLEBUF);
```

Para terminar usamos a seguinte função:

```
SDL_Quit();
```

De forma a compreendermos melhor, encontra-se no anexo A.5 Código-fonte da visualização onde se pode ver com mais detalhe o modo de implementação da biblioteca SDL.

As resoluções de vídeo suportadas foram as seguintes:

- 1280x704p 16:9 -50Hz,25 Hz
- 640x352p 25 Hz, 12.5Hz
- 320x176p 25 Hz, 12.5Hz

Respetivamente HD, SD e CIF. As resoluções inferiores são obtidas por subamostragem das resoluções superiores, metade na horizontal e metade na vertical.

4.6 Cenários de teste e discussão de resultados

Para se entender os cenários em que o terminal de TV foi testado e os resultados, é necessário compreender a localização dos emissores e suas características, bem como do local onde os mesmos foram efetuados.

Figura 40 - Localização dos emissores.

Os emissores de DVB-T/RCT e WiMAX foram instalados em duas localizações diferentes, confome se mostra na Figura 40. Um dos locais onde foi instalado o emissor fica no prédio mais alto de Aveiro, o prédio da Segurança Social, iremos referir este prédio como SS, latitude 40°38'38.75"N e longitude 8°38'57.25"W. O outro local onde o emissor foi instalados foi no prédio do Instituto de Telecomunicações, iremos referir este prédio como IT, latitude 40°38'1.49"N e longitude 8°39'35.23"W. Cada local tinha ambos emissores e estava equipado com duas antenas UHF (uma para emissão e outra para receção). O emissor DVB-T/RCT emitia no canal 43 (frequência 650 MHz) e recebia no canal 59 (frequência 778 MHz). A potência radiada pelo emissor que se encontrava no prédio do IT era de, aproximadamente, 200 W enquanto o emissor que se encontrava no prédio da Segurança Social era de, aproximadamente, 100 W. Relativamente aos emissores de WiMAX, o que se encontrava no IT operava na frequência 2515 MHz com uma potência à saída do amplificador de, aproximadamente, 1,5 W e o que se encontrava na SS operava na frequência 2535MHz com uma à saída do amplificador de 0,50 W. Devido à natureza da mobilidade todas as antenas tinham polarização vertical. Ficando o *playout* no IT, foi então instalado um *mini-link* entre o prédio do IT e da Segurança Social. A distância entre eles é de, aproximadamente, 1,5 quilómetros.

Os testes foram efetuados na autoestrada A25, campo aberto, e no centro da cidade de Aveiro, ambiente urbano, como mostram as seguintes figuras Figura 41 e Figura 42, respetivamente.

Figura 41 - Percurso na autoestrada A25 emissor DVB-T/RCT IT.

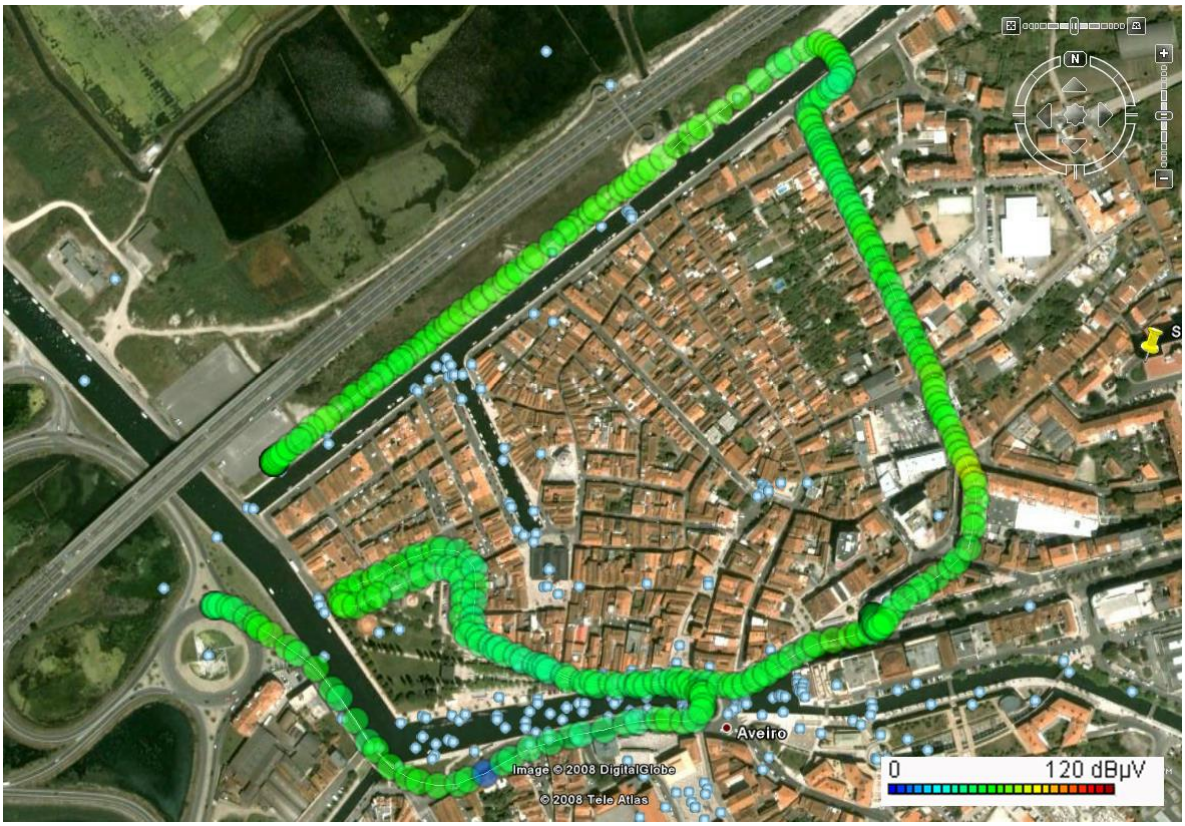


Figura 42 - Percurso na cidade Aveiro emissor DVB-T/RCT SS.

O teste com o terminal parado foi efetuado em frente ao IT, também fora efetuados testes no laboratório do IT sem os emissores, apenas com os cabos de rede.
Para analisar os diversos cenários em que o terminal de TV iria ser posto à prova, foi necessário fazer uma análise de vários parâmetros do canal de comunicação, entre eles, largura de banda, *jitter* e perda de pacotes de forma a avaliar a performance da rede. Para isso foram usados dois *softwares* o iperf e o wireshark. Os resultados obtidos serão apresentados em gráficos, nas figuras seguintes.

Todos os testes foram efetuados usando uma modulação para o emissor DVB-T/RCT 16QAM 2/3 e para o emissor WiMAX uma modulação adaptativa, o que permite uma largura de banda máxima de 14 Mbps e 8 Mbps, respetivamente, no *downlink e* 2 Mbps no *uplink* [23].

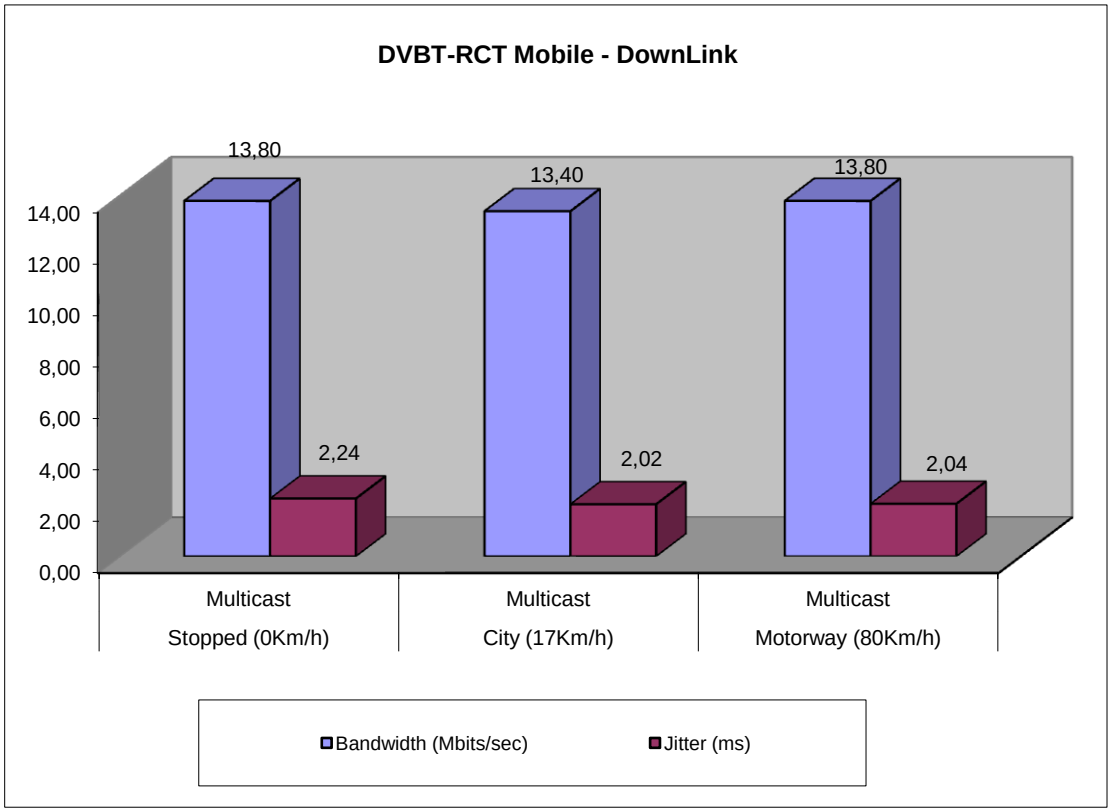


Figura 43 - Largura de banda e *Jitter* em DVB-T/RCT *downlink.*

Pela análise da Figura 43 podemos concluir que nos três cenários o DVB-T/RCT tem uma performance estável, sendo capaz de enviar dados até 13.8 Mbps e, com esta largura de banda é possível enviar até dois serviços HD.

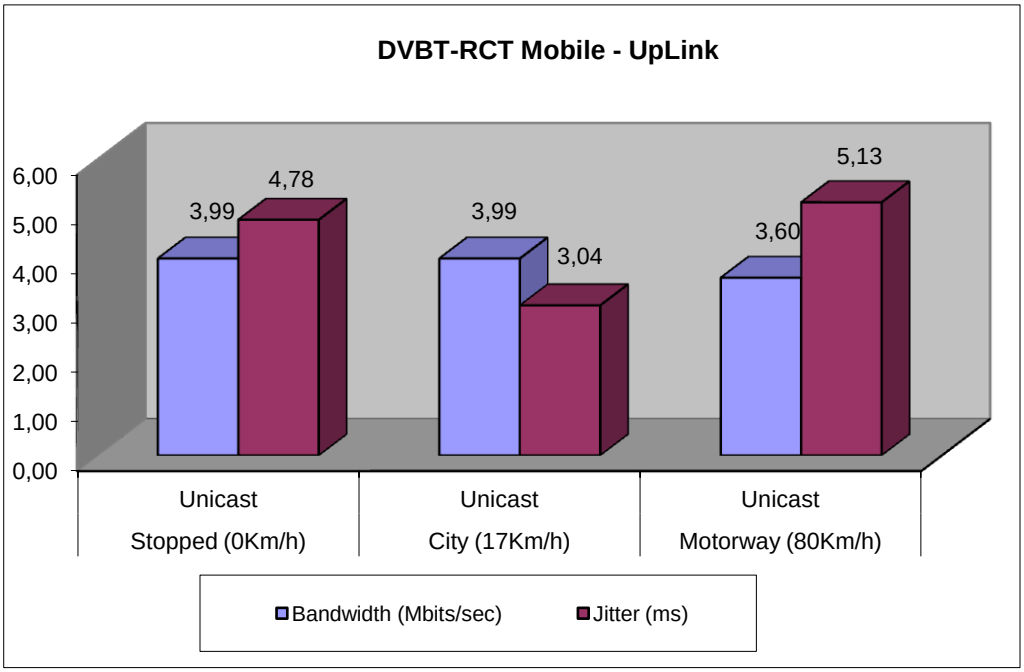


Figura 44 - Largura de banda e *Jitter* em DVB-T/RCT *uplink.*

Na Figura 44 podemos verificar que o *jitter* é muito elevado o que pode indicar que o canal de comunicação é vulnerável a multipercurso e ao efeito *Doppler*. Muitos dos problemas surgem de o receptor não esta em linha de vista com a antena emissora, existe a contribuicao de alguns ecos (adicao ruido gaussiano), resultando num canal multipercurso (*Rayleigh*) que tem grande variacao com o tempo, a frequencia e o local.Adicionalmente, no caso de uma receptcao movel, o desvio de frequencia devido a cada eco recebido (Efeito *Doppler*), varia com a velocidade e com a direccao. Esta dispersao de ecos provoca interferencia entre simbolos, porque vai contra a lei da ortogonalidade de sinais COFDM. Por outro lado temos o efeito de Doppler afectado pelos ecos recebidos.A largura de banda máxima é 3.9 Mbps que pode ser utilizada para comunicações VoIP, internet e outros serviços de dados.

No DVB-T/RCT *downlink* apenas foram efetuados testes em *multicast*, ou seja, serviços em tempo real porque os serviços VoD, em *unicast*, apenas serão difundidos pelos WiMAX.

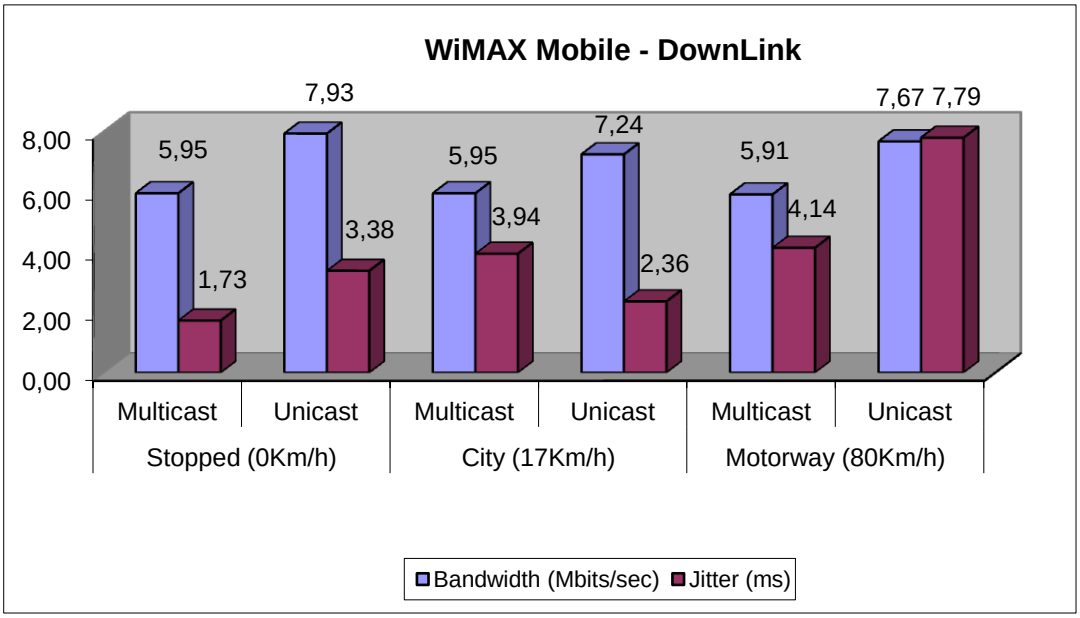


Figura 45 - Largura de banda e *Jitter* em WiMAX *downlink.*

Nos resultados apresentados na Figura 45 pode-se verificar que a largura de banda nos dois cenários, *multicast* e *unicast* do WiMax *downlink*, é praticamente a mesma, sendo respetivamente 5.9 Mbps e 7.6 Mbps. Com esta largura de banda é possível oferecer o serviço de VoD, VoIP e internet. Observando o *jitter*, verifica-se que na autoestrada o valor é duas vezes superior à situação parado, isto pode dever-se à velocidade e consequentemente ao efeito *Doppler*.

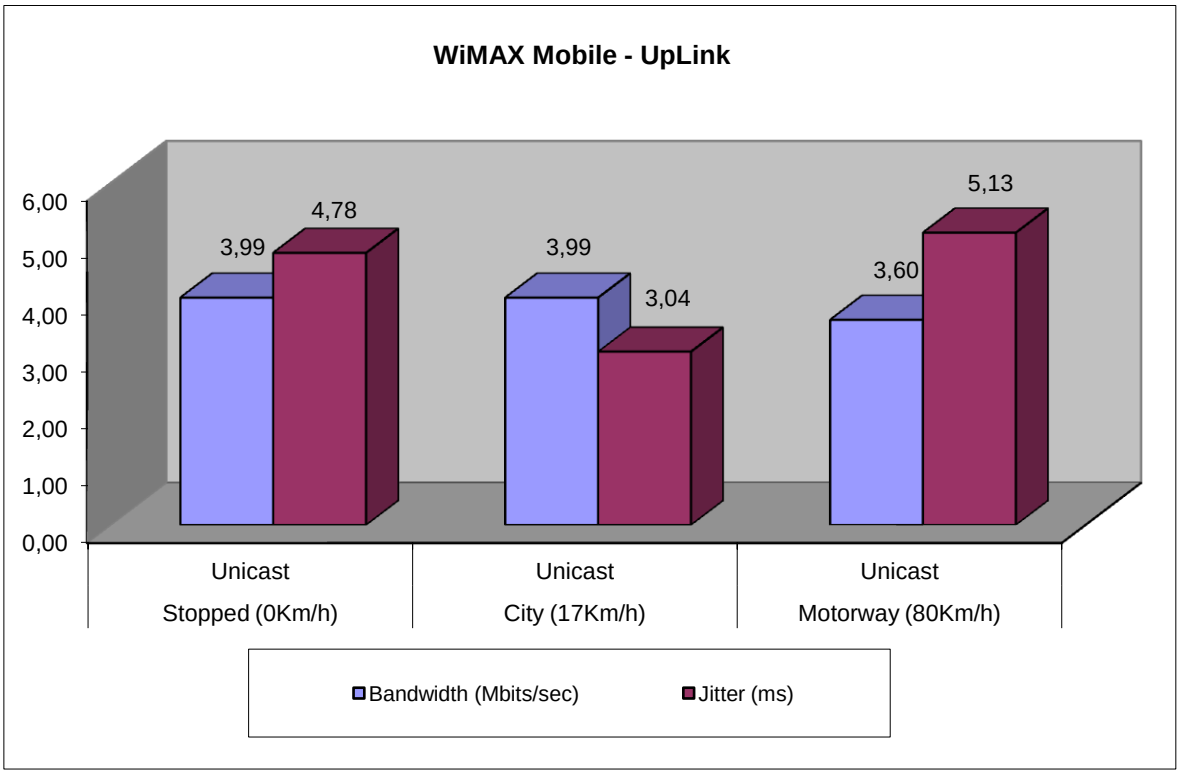


Figura 46 - Largura de banda e *Jitter* em WiMAX *uplink.*

Do gráfico da Figura 46 pode-se verificar que a largura de banda no WiMAX uplink é de 4 Mbps, apenas na autoestrada ligeiramente inferior. Mais uma vez, conclui-se que devido ao valor elevado do *jitter* o WiMAX uplink sofre de multipercurso e do efeito de *Doppler*. Toda a análise da largura de banda e do *jitter* foi feito através do *software* iperf.

Agora vai-se analisar a perda de pacotes na rede, nas diferentes situações, igualmente descritas em cima. Numa primeira fase será feita uma análise à perda de pacotes UDP, referente à camada de transporte usando o *software* iperf e posteriormente aos pacotes RTP, referente à camada de aplicação, usando o *software* wireshark. O tempo de captura de pacotes foi muito idêntico em todas as situações.

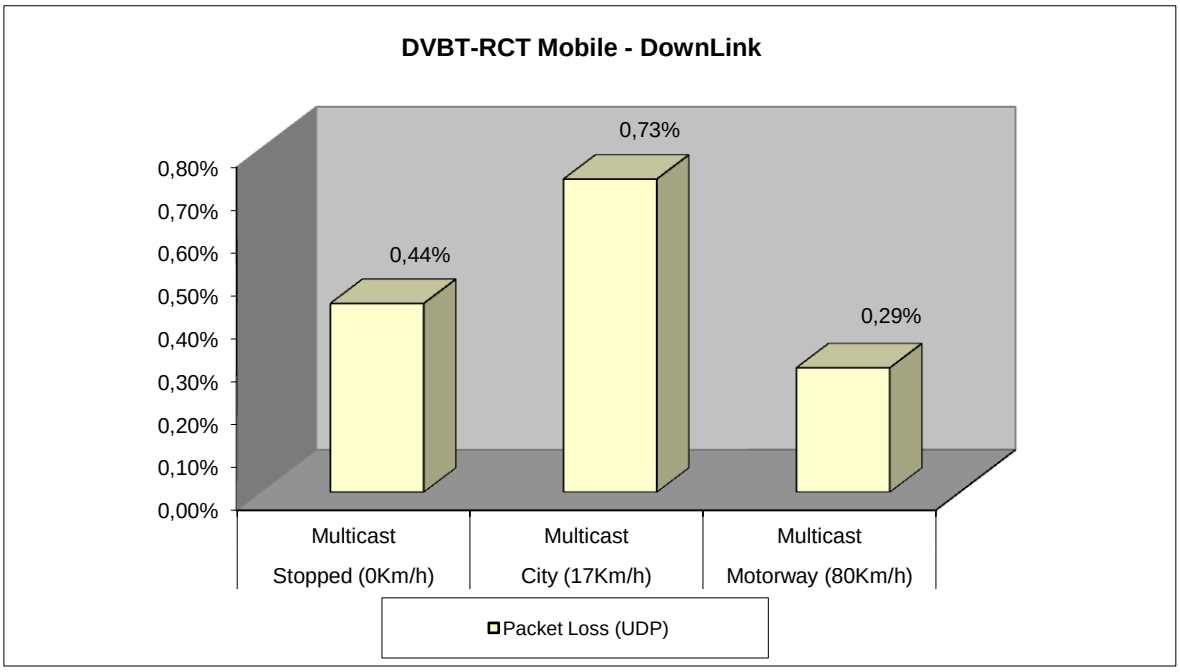


Figura 47 - Perda de pacotes UDP em DVB-T/RCT *downlink.*

Os valores da perda de pacotes UDP *multicast* em DVB-T/RCT *downlink* na Figura 47 estão de acordo com o esperado, contudo o cenário parado deveria ter uma perda inferior ao da autoestrada.

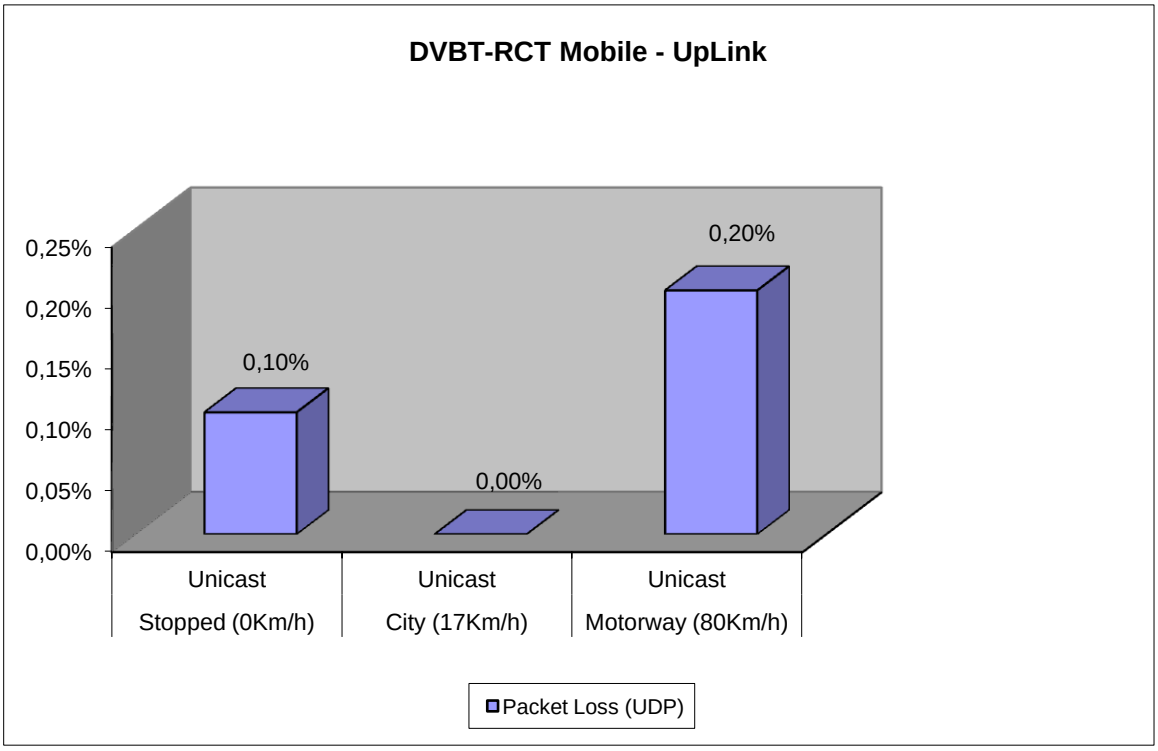


Figura 48 - Perda de pacotes UDP em DVB-T/RCT *uplink.*

Como pode ser observado pela Figura 48 o canal de retorno do DVB-T/RCT têm uma perda de pacotes UDP muito baixa.

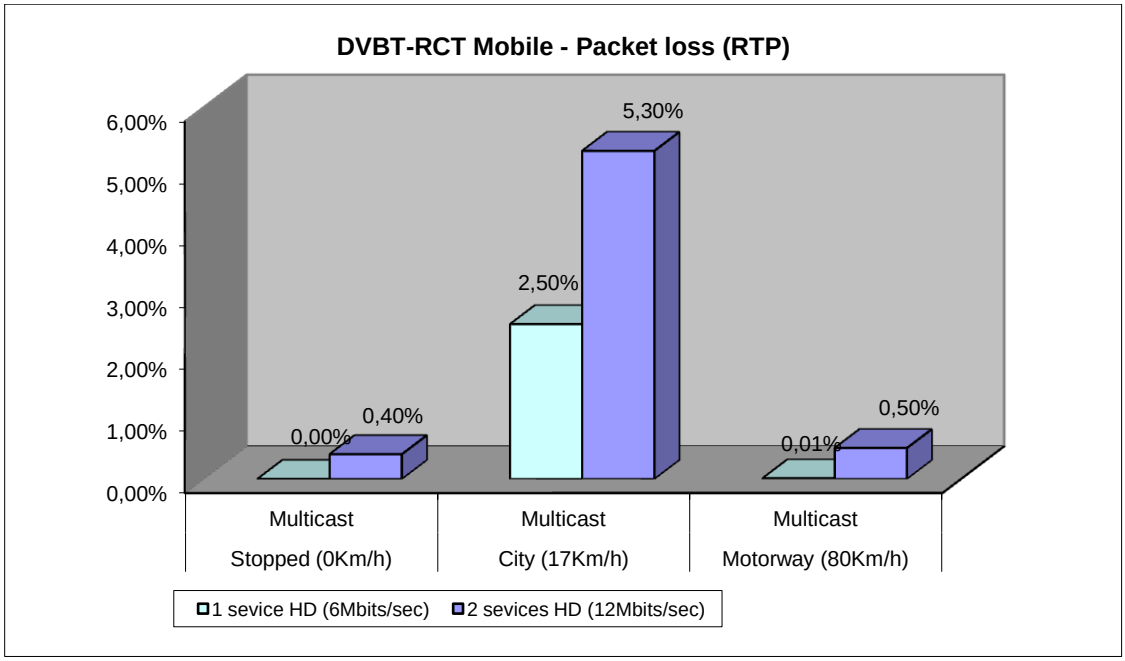


Figura 49 - Perda de pacotes RTP DVB-T/RCT com um ou dois serviços HD.

Os valores da Figura 49 apresentam a perda de pacotes RTP no DVB-T/RCT com um serviço HD ou com dois serviços HD. Pode-se observar o aumento da perda de pacotes RTP quando estão dois serviços HD a serem fornecidos, comparativamente com um serviço HD. Os valores estão de acordo com o previsto e de acordo com a perda de pacotes UDP.

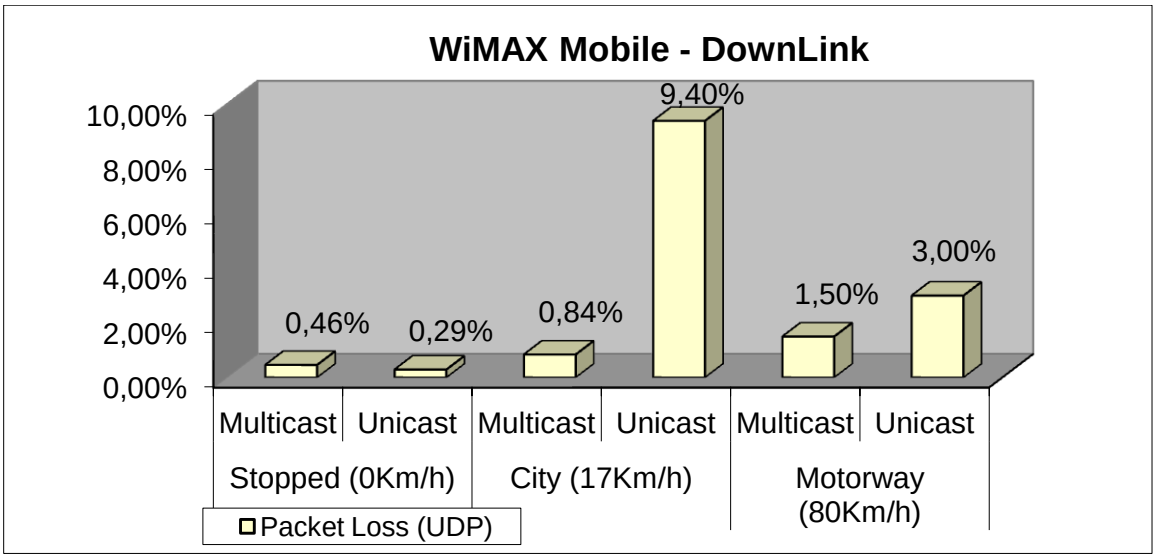


Figura 50 - Perda de pacotes UDP WiMAX *downlink.*

A Figura 50 mostra a perda de pacotes RTP no canal de *downlink* do WiMax. Os resultados demonstram que o sistema tem uma boa performance em *multicast* bem como em *unicast*, porém no cenário da cidade o sistema é mais sensível à modulação adaptativa. Em zonas onde existem vários obstáculos a modulação baixa um ou dois níveis de acordo com o *carrier to interference plus noise ratio* (CINR) que tem no momento, no *multicast* isto não é tão visível.

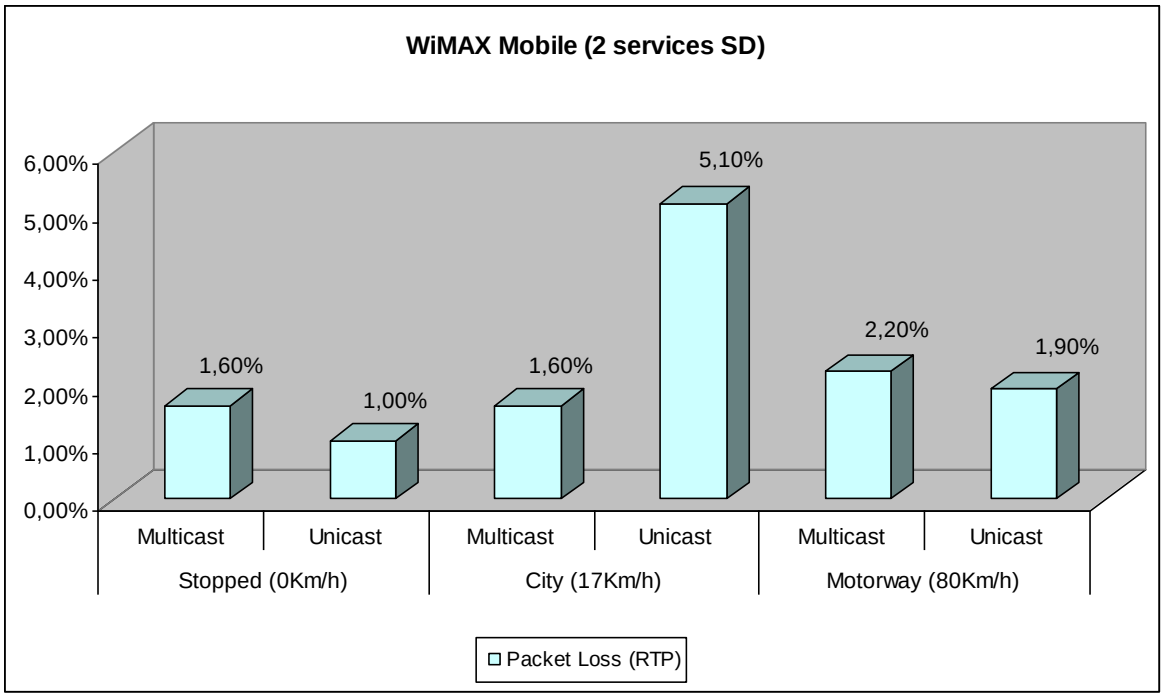


Figura 51 - Perda de pacotes RTP em WiMAX com dois serviços SD.

A Figura 51 apresenta a perda de pacotes RTP em WiMAX com dois serviços SD de forma a utilizar a rede nas máximas condições. Comparando com a Figura 50 pode-se concluir que a perda de pacotes é muito idêntica e que no caso do *unicast* na cidade, a perda de pacotes é relativamente elevada, devido ao terminal de TV não esta em linha de vista com a antena emissora, a contribuicao de alguns ecos (adicao ruido gaussiano), resultando num canal multipercurso (*Rayleigh*) que tem grande variacao com o tempo, a frequencia e o local. O *uplink* do WiMAX também apresentou perda de pacotes UDP elevados, do que se pode concluir que o sistema DVB-T/RCT apresenta melhores resultados no que diz respeito à perda de pacotes UDP e RTP.

O atraso na rede medido no DVB-T/RCT e WiMAX foi de, aproximadamente, 23 ms e 60 ms, respetivamente.

Foi também realizado um teste no laboratório com os cabos de rede ligados, ou seja, sem a interface RF e o mesmo teste mas usando a interface RF, incluindo o *multiplexer* (*gateway*).

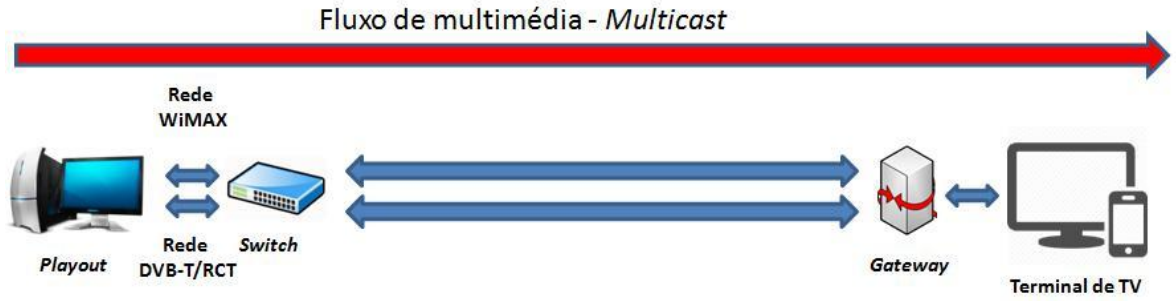


Figura 52 - Teste do terminal usando cabos de rede.

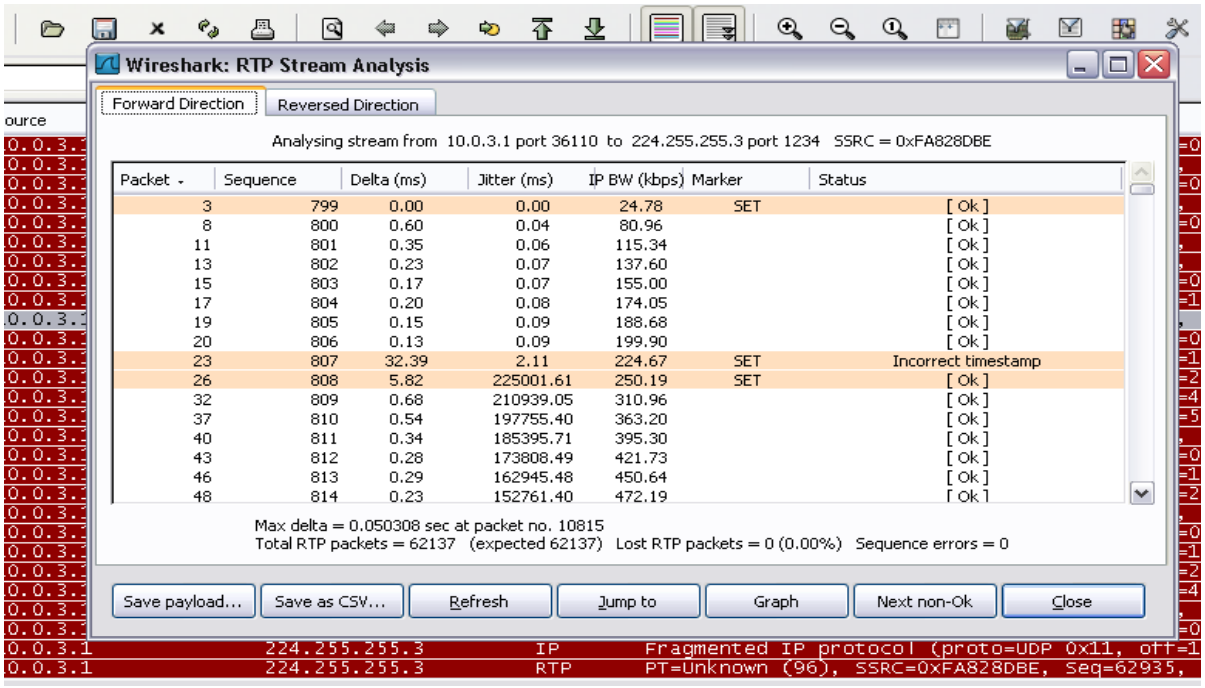


Figura 53 - Perda de pacotes RTP com um service HD usando cabos de rede.

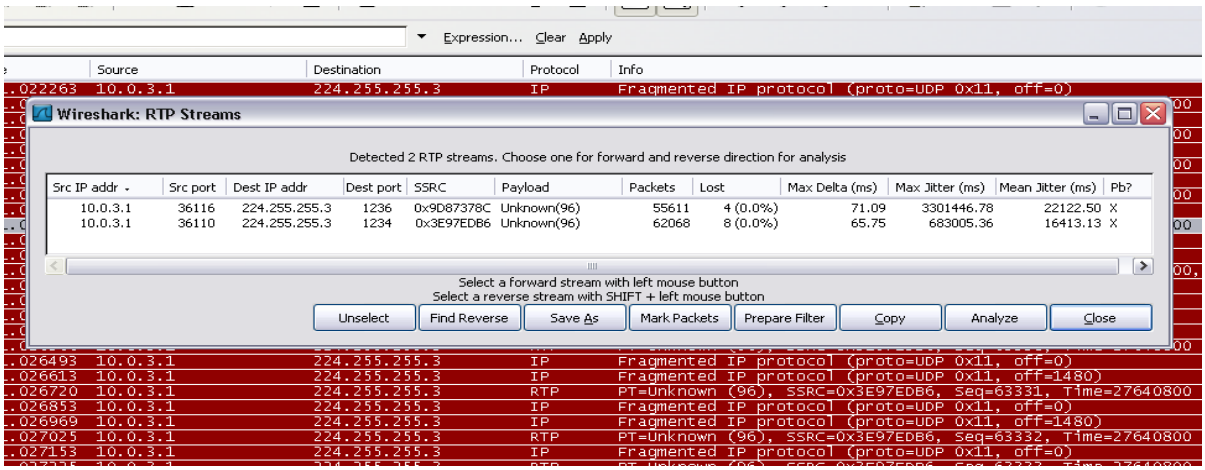


Figura 54 - Perda de pacotes RTP com dois serviços HD usando cabos de rede.

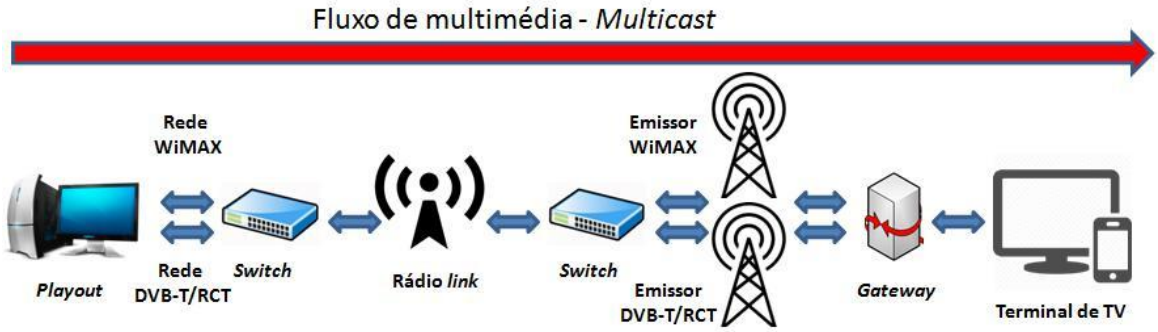


Figura 55 - Teste do terminal usando a interface RF.

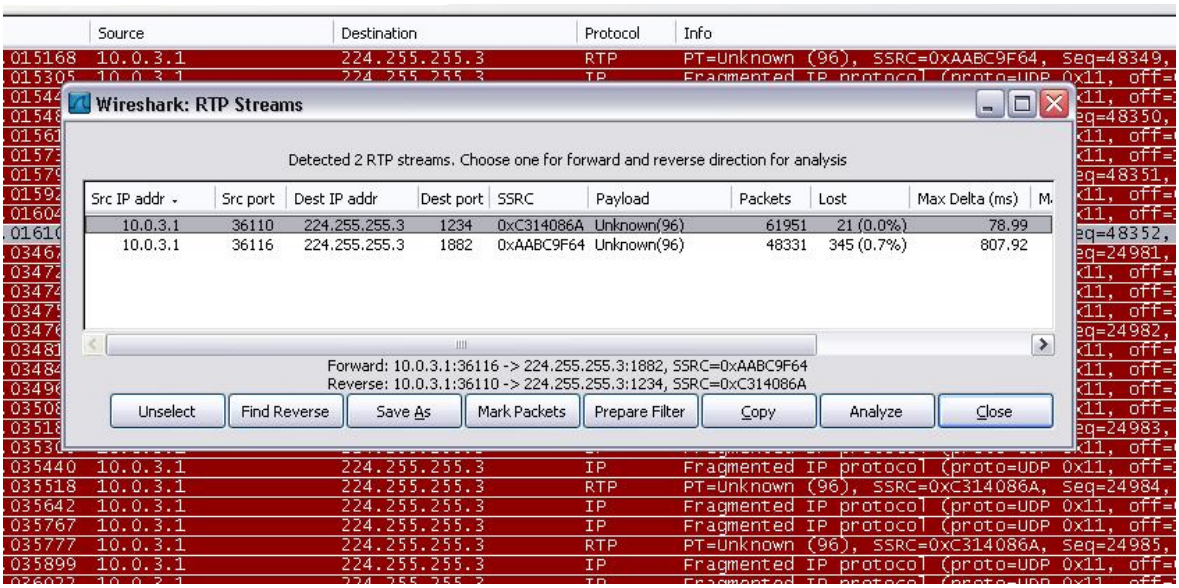


Figura 56 - Perda de pacotes RTP com dois serviços HD usando a interface RF.

Na Figura 52 podemos ver um diagrama do teste realizado em laboratório usando cabos de rede *ethernet* e na Figura 53 os resultados dos mesmos no terminal de TV usando o *software wireshark*. A perda de pacotes RTP foi de 0 %.

Seguidamente, realizou-se um teste usando a interface RF, como podemos ver no diagrama da Figura 55. O resultado obtido, usando o mesmo *software*, para a perda de pacotes RTP foi de 0 % para a descrição D1 e de 0,7 % para a descrição D2.

De referir que neste caso a *gateway* foi transparente para o terminal de TV, ou seja, não multiplexando as duas descrições, sendo que a descrição D1 vai por DVB-T/RCT e a descrição D2 vai pelo WiMAX.

Em conclusão, pelas figuras apresentadas anteriormente, podemos verificar que a perda de pacotes é muito baixa, apesar disso, quando o terminal de TV se encontra no cenário móvel dentro da cidade a perda de pacotes já é suficientemente elevada. Foi esta uma das razões

que levou ao desenvolvimento de um algoritmo robusto de cancelamento de imagem, descrito anteriormente e mostrado na Figura 37. Após estes resultados, realizou-se testes em tempo real, com uma câmara HD, visualização de *websites*, chamadas VoIP e serviços VoD com sucesso.

Devido à versatilidade do terminal, foi possível simular os diferentes formatos CIF,SD e HD em *multicast* e *unicast*, fixo ou móvel.

Verificou-se, ainda, que a largura de banda utilizada em *multicast* não depende do número de terminais de TV. Já em *unicast* a largura de banda utilizada varia consoante o número de terminais de TV existentes.

CAPÍTULO 5 – Conclusões e trabalho futuro

Relativamente ao trabalho efetuado, os requisitos que foram formulados no início do projeto para o terminal de TV foram executados com sucesso, nomeadamente:

- ✓ A reprodução de uma emissão em direto, que é codificada em tempo real pelo *playout*;
- ✓ A escalabilidade do vídeo nos diferentes formatos (CIF, SD e HD);
- ✓ A aplicação em ambiente gráfico;
- ✓ VoD com a função *pause* e *play*;
- ✓ Um *browser* para a visualização de páginas *web*;
- ✓ A inclusão do Skype para efetuar comunicações VoIP.

5.1 Principais conclusões

A escalabilidade é inevitável, apesar do histórico não mostrar isso. Para que os terminais de TV, sejam eles, móveis ou fixos, com diferentes resoluções e capacidades de descodificação, possam ser cada vez mais utilizados, é fundamental usar os recursos que estão disponíveis. O uso das comunicações *multicast* revelou-se de elevada importância e eficaz mas ao mesmo tempo difícil e lenta de implementação.

Graças à escalabilidade do vídeo, nas zonas em que existe uma limitação de largura de banda por razões que são impostas pela rede (não haver cobertura HD ou largura de banda suficiente) ou pelo número de utilizadores a utilizarem o serviço VoD, o terminal de TV consegue adaptar-se de forma a comutar automaticamente entre os diferentes formatos (CIF,SD e HD), preferindo sempre a resolução HD caso esta esteja disponível.

A conjugação única dos sistemas de transmissão DVB-T/RCT e WiMAX, permitiu uma ampla cobertura mesmo nas zonas de mais difícil acesso. Os ganhos resultantes desta integração são claros. O algoritmo implementado no terminal de TV, permite uma transição entre os dois modos de transmissão sem que o utilizador se aperceba desse facto. Não só apresenta esta mais-valia como permite que a perda de pacotes seja muito menor e assim que existam menos paragens (*freezing*) na imagem. O utilizador só se apercebe caso a passagem seja de uma resolução HD para CIF.

A escolha de codificação pelo *playout* foi essencial para que, numa eventualidade de falta de sinal e posterior recuperação, o terminal de TV consiga apresentar sempre uma imagem completa, ou seja, sempre que exista perda de pacotes e não seja possível reconstruir a imagem esta é descartada até à próxima imagem completa. Isto foi possível porque o *playout* codificou o vídeo usando apenas imagens I e P e com um GOP de 1 segundo.

Hoje em dia assiste-se a uma tendência para a convergência em tudo o que é digital, os crescentes serviços e funcionalidades agregam-se em números mais reduzidos de equipamentos, o conceito de *"Anything, Anywhere, Anytime"* é uma máxima que reflete esse progresso.

Os testes descritos na secção 4.6 mostraram que se conseguir receber no terminal TV um débito, máximo, de 14 Mbps de *download* e 2 Mbps de *upload* por DVB-T/RCT e de 8 Mbps de *download* e 2 Mbps de *upload* por WiMAX. Apesar das perdas de pacotes, o terminal de TV desenvolvido mostrou-se muito robusto.

O terminal de TV utiliza um processador VIA C7 1.8 GHz com 1GB de RAM que se mostrou suficiente para decodificar SVC até HD (720p), usando uma placa gráfica NVIDIA GeForce 8400 GS. O tempo máximo ocupado pelo processador foi de 15 %.

Finalmente, de referir que a partir desta dissertação de mestrado foi elaborado um artigo científico que foi apresentado no simpósio Internacional do IEEE, no Algarve [24], galardoado com o 3º melhor artigo científico da conferência.

5.2 Trabalho futuro

Dizer que se tem uma solução definitiva é sinónimo de não ter percebido a extensão do domínio de aplicação aqui envolvido, certamente muito mais haveria a acrescentar, muitas mais teses se escreveriam sobre esta matéria e os caminhos continuarão abertos, até mais, se é possível.

No que diz respeito ao *hardware*, o terminal de TV está a correr no *setup* mínimo para que seja possível a descodificação de uma transmissão HD, no entanto, usando novas rotinas que usem de forma mais eficiente o GPU da placa gráfica ou recorrendo a um processador dedicado para a descodificação de vídeo iria permitir com que o software do terminal de TV funcionasse da mesma forma num *setup* menos exigente em termos de hardware (CPU).

Em termos de interface, poder-se-ia ter usado outras fontes e disposição dos botões bem como da lista de canais de forma a melhorar o *design*.

A inclusão de algoritmos de criptografia nas comunicações entre o *playout* e o terminal de TV, novas formas de autenticações dos utilizadores no grupo *multicast*, seria uma mais-valia.

Seria também importante a introdução do EPG, que daria um guia diário ou semanal, consoante a escolha, das horas a que os programas podem ser televisionados e permitindo deste modo ao utilizador, sem estar em casa, gravar os programas que gostava de ver, tendo para isso que programar o terminal de TV antes de sair de casa.

Relativamente à apresentação da imagem, a introdução de um *picture in picture* (PIP), personalizável pelo utilizador, para que possa ver em simultâneo dois canais e/ou saber o que se passa no canal vizinho, bem como o *hyperlinked video* que introduz uma inovação tecnológica no terminal de TV, que possibilita a visualização de um acontecimento passado, por exemplo, num jogo de futebol ter a possibilidade de rever um golo.

A ser implementado, quer esta nova funcionalidade como o PIP, deve-se fazer um pedido RTSP ao servidor para que este lhe forneça, ao terminal de TV que fez o pedido, o conteúdo no formato CIF.

REFERÊNCIAS

[1] *SUIT-166, D1.4*: Architecture and Reference scenarios – FP6-IST SUIT Project, December 2006.

[2] T. Wiegand, G. J. Sullivan, J. Reichel, H. Schwarz, and M. Wien, Joint Draft 7 of SVC Amendment (revision 2), Klagenfurt, Austria, JVT-T201, 15–21 July, 2006.

[3] Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television. ETSI, 01/2009.

[4] Air Interface for Fixed and Mobile Broadband Wireless Access Systems, IEEE 802.16e – 2005, December 2005.

[5] Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), ITU-T and ISO/IEC JTC 1, Version 1: May 2003, Version 2: May 2004, Version 3: Mar. 2005, Version 4: Sept. 2005, Version 5 and Version 6: June 2006, Version 7: Apr. 2007, Version 8 (including SVC extension): Consented in July 2007.

[6] Digital Video Broadcasting (DVB); Interaction channel for Digital Terrestrial Television (RCT) incorporating Multiple Access OFDM, ETSI EN 301 958, 03/2002.

[7] Handley, M., Jacobson, V., and Perkins, C., Jul. 2006. SDP: Session Description Protocol, IETF, Request for Comments, RFC 4566.

[8] Handley, M., Perkins, C., and Whelan, E., 2000. Session Announcement Protocol. IETF, Request for Comments, RFC 2974.

[9] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E., "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[10] Schulzrinne, H., Rao, A., and Lanphier, R., 1998. Real Time Streaming Protocol (RTSP). IETF, Request for Comments, RFC 2326.

[11] Bormann, K.O., Feb. 2005. Session Description and Capability Negotiation. IETF, Internet Draft, draft-ietf-mmusic-sdpng-08.

[12] Digital Video Broadcasting (DVB); Transport of MPEG-2 Based DVB Services over IP Based Networks, 2005, ETSI TS 102 034 V1.1.1.

[13] Digital Video Broadcasting (DVB); IP Datacast: Program Specific Information (PSI)/Service Information (SI); Part 1: IP Datacast over DVB-H, 2009, ETSI TS 102 470-1 V1.2.1.

[14] Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems, 2016, ETSI EN 300 468 V1.15.1.

[15] Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP), Specification 1.0.3, 2003, ETSI TS 101 812 V1.3.1.

[16] A DNS RR for specifying the location of services (DNS SRV), IETF, RFC 2782

[17] Fenner, W., 1997. Internet Group Management Protocol. IETF, Request for Comments, RFC 2236.

[18] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and Thyagarajan, A., 2002. Internet Group Management Protocol, Version 3. IETF, Request for Comments, RFC 3376.

[19] MPEG MDS Group, Multimedia framework Part 7: Digital item adaptation, Final Draft International Standard, Doc. ISO/MPEG, N6168, MPEG Waikaloa Meeting, USA, December 2003.

[20] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V., Jul. 2003. RTP: A Transport Protocol for Real-Time Applications. IETF, Request for Comments, RFC 3550.

[21] Wenger, S., Hannuksela, M.M., Stockhammer, T., Westerlund, M., and Singer, D., Feb. 2005. RTP Payload Format for H.264 Video. IETF, Request for Comments, RFC 3984.

[22] *SUIT-213, D4.1*: Synchronization and Encapsulation – FP6-IST SUIT Project, February 2006.

[23] *SUIT-520, D6.5*: Analysis of Results– FP6-IST SUIT Project, June 2008.

[24] Nuno Coelho, Nelson Cabral, David Marques, Antonio Navarro, Mobile HDTV at 140 Km/h, 2008. ISCE 2008. ISCE - IEEE International Symposium on Consumer Electronics Algarve, Portugal

Anexos

A.1 Exemplo de SDP

```
v=0
o=- 6645483124 1 IN IP4 10.0.2.1
s=Streamed by SUIT Playout Server
i=VoD_Fighter
t=0 0
a=tool:LIVE555 Streaming Media v2008.02.08
a=type:broadcast
a=control:*
a=x-qt-text-nam:Streamed by SUIT Playout Server
a=x-qt-text-inf:VoD_Fighter
m=video 0 RTP/AVP 96
c=IN IP4 0.0.0.0
a=rtpmap:96 H264/90000
a=fmtp:96 packetization-mode=1;profile-level-id=4D4020;sprop-parameter-
sets=J01AIJZWCgz8+AiAABOIAAPQkHNgAnqAJ6zOfAQA,KM48gAA=
a=control:track1
m=audio 0 RTP/AVP 14
c=IN IP4 0.0.0.0
a=range:npt=0-145.604
a=control:track2
```

A.2 Ficheiro rcic.ini

```
RCIC_ADDRESS 10.0.1.1
RCIC_PORT 20000
TIMEOUT 5
LOGIN_NAME nelson
SESSION_ID Session02
RES_HOR 1280
RES_VER 704
FRAME_RATE 25
PASSWORD nelson
```

A.3 Código-fonte principal da aplicação

```
    AVCodec *codec;
    AVCodecContext* decoder;
    AVFormatContext* formatContext;
    AVFrame *recFrame;
    AVInputFormat* fmt = &h264_demuxer;
    AVPacket packet;//data packet which will be filled by the bitstream
    YuvImage image;
    HRenderer renderer = NULL;
    LONGLONG start_time = 0;    //for displaying at the right frame rate
    int frame_number = 0;
    int frame_interval;

    FILE* fstream = NULL;
    FILE* fyuv = NULL;
    int nalu_cnt = 0;
    int ret = 0;
    fmt->flags = AVFMT_NOFILE; //Do not try to open a file (RTP hack
Cf.av_open_input_file(), must_open_file)

    //RTP stuff
      WSADATA wsaData;
      WORD wVersionRequested = MAKEWORD( 2, 2 );
      WSAStartup(wVersionRequested, &wsaData); //WSACleanup called in
OnDestroy
    c_ProcessRTPSession rtpSession;

    // parse options
    parse_options(argc, argv, options);

    if(!ip)
    {
        printf("Bad multicast IP: %s\n", ip);
        syntax();
        exit(1);
    }

    if(!port)
    {
        printf("Bad multicast port: %d\n", port);
        syntax();
        exit(1);
    }

    if (output_file)
    {
        fyuv = fopen(output_file,"wb");
        if (fyuv == NULL)
        {
            printf("Unable to open %s\n",output_file);
            syntax();
            exit(1);
        }
    }

    rtpSession.Initialize();
    RunRtpClient(rtpSession);
```

```
    // Register all formats and codecs
    av_register_all();

#ifdef _DEBUG
    av_log_set_level(AV_LOG_DEBUG);
#endif

    //allocate decoded frame
    recFrame = avcodec_alloc_frame();

    //allocate FormatContext
    ret = av_open_input_file( &formatContext, NULL/*INPUT_FILE*/, fmt, 0,
NULL );
    if(ret)
    {
        printf("Unable to open the RTP stream at: %s port %d\n", ip,
port);
        exit(1);
    }

    //Initialize buffers
    int buffer_size = MAX_AU_SIZE;
    uint8_t *buffer = (uint8_t*)av_malloc(buffer_size);

    init_put_byte(&formatContext->pb, buffer, buffer_size, 0/*flag*/,
&rtpSession/*opaque*/, rtp_read_packet/*read_packet*/,
NULL/*write_packet*/, NULL);

    //parse SPS, PPS and fill streams
    av_find_stream_info( formatContext );
    frame_interval = 1000*formatContext->streams[0]->codec-
>time_base.num/formatContext->streams[0]->codec->time_base.den;

    // We decode only video stream so first stream is video stream
    decoder = formatContext->streams[0]->codec;

   //Get H264 codec
    codec = avcodec_find_decoder(decoder->codec_id);

    //Open Codec
    if(avcodec_open(decoder, codec)<0)
    {
        printf("Unable to open codec %s\n",argv[1]);
        exit(1);
    }

    spatial = FFMIN(decoder->dependency_layers - 1 , spatial);
    quality = FFMIN(decoder->quality_layers - 1 , quality);

    //create renderer
    if( RDR_create( &renderer, decoder->width_layer[spatial], decoder-
>height_layer[spatial], fyuv ) < 0 )
    {
        printf("Unable to open codec %s\n",argv[1]);
        exit(1);
    }

    decoder->dqid_plus1 = ((spatial << 4) + quality) + 1 ;
```

```
    //MAIN LOOP
    while( av_read_frame( formatContext, &packet ) >= 0 )
    {
        if( packet.stream_index == 0 )
        {
          int got_picture;

          avcodec_decode_video( decoder,
                                recFrame, &got_picture,
                                packet.data, packet.size );


          WaitForNextFrameToRender(start_time, frame_number,
frame_interval);

          if(got_picture)
          {


            if(    ( decoder->width  != decoder->width_layer[spatial]  )
                || ( decoder->height != decoder->height_layer[spatial] )
){
                AVPicture picture;
                struct SwsContext *resample_ctx;

                avpicture_alloc(&picture, PIX_FMT_YUV420P,
                                decoder->width_layer[spatial], decoder-
>height_layer[spatial]);

                resample_ctx = sws_getContext(decoder->width, decoder-
>height, PIX_FMT_YUV420P,
                                              decoder-
>width_layer[spatial], decoder->height_layer[spatial], PIX_FMT_YUV420P,
                                              SWS_FAST_BILINEAR, NULL,
NULL, NULL);


                sws_scale(resample_ctx, recFrame->data, recFrame-
>linesize,
                          decoder->width, decoder->height, picture.data,
picture.linesize);

                sws_freeContext(resample_ctx);

                image.plane[0].addr = picture.data[0];
                image.plane[1].addr = picture.data[1];
                image.plane[2].addr = picture.data[2];

                image.plane[0].stride = picture.linesize[0];
                image.plane[1].stride = picture.linesize[1];
                image.plane[2].stride = picture.linesize[2];


                image.plane[0].width =  decoder->width_layer[spatial];
                image.plane[1].width =
                image.plane[2].width =  decoder->width_layer[spatial] /
2;

                image.plane[0].height = decoder->height_layer[spatial];
```

```
                image.plane[1].height =
                image.plane[2].height = decoder->height_layer[spatial] /
2;

                RDR_blit( renderer, &image );

                avpicture_free(&picture);

            } else {

                image.plane[0].addr = recFrame->data[0];
                image.plane[1].addr = recFrame->data[1];
                image.plane[2].addr = recFrame->data[2];


                image.plane[0].stride = recFrame->linesize[0];
                image.plane[1].stride = recFrame->linesize[1];
                image.plane[2].stride = recFrame->linesize[2];


                image.plane[0].width =  decoder->width;
                image.plane[1].width =
                image.plane[2].width =  decoder->width / 2;

                image.plane[0].height = decoder->height;
                image.plane[1].height =
                image.plane[2].height = decoder->height / 2;

                RDR_blit( renderer, &image );
            }

          }
          frame_number++;
        }
        av_free_packet(&packet);
        RDR_pollEvent(renderer);

    }

    //RTP stuff
    av_free(buffer);
    rtpSession.UnInitialize();
    //leave the multicast
    //FIXME
    //rtpSession.LeaveMulticastGroup(rtpAddr);

    //destroy the session
      rtpSession.Destroy();
      WSACleanup();

    RDR_delete( renderer );

    avcodec_close(decoder);


    if (fstream != NULL)
        fclose(fstream);
```

A.4 Código-fonte do desencapsulador

```
// SUIT
typedef struct
{
    // specific to SUIT project
    int i_actual_type;
    // specific to SUIT project

    int i_simple_priority_id;       // PRID 6 bits
    int i_temporal_level;           // TL   3 bits
    int i_dependency_id;            // DID  3 bits
    int i_quality_level;            // QL   2 bits
    int i_layer_base_flag;          // B    1 bit
    int i_use_base_prediction_flag; // U    1 bit
    int i_discardable_flag;         // D    1 bit
    int i_fragmented_flag;          // G    1 bit
    int i_last_fragmented_flag;     // L    1 bit
    int i_fragment_order;           // O    2 bit
} suit_nal_extension_t;
static int nbSpatialLayers = 0;
static int nbSnrLayers = 0;

typedef enum
{
    STATENULL = 0,
    STATE0 = 1,
    STATE00 = 2,
    STATE000 = 3,
    STATE0001 = 4
} StartCodeSM;

/*

  findNextStartCode read the FILE parameter until it reach a start code
(three '0' and one '1')
  It then returns the nal header and the following 3 bytes.
  These 4 bytes are the nal header + nal header extension in the case of
SVC.

*/

void findNextStartCode(FILE* file, unsigned char* nal_header)
{
    StartCodeSM state;
    state = STATENULL;
    while(state != STATE0001 && !feof(file))
    {
        char c = fgetc(file);
        switch(state)
        {
        case STATENULL:
            if (c==0) state = STATE0;
            break;
        case STATE0:
            if (c==0) state = STATE00;
            else state = STATENULL;
            break;
        case STATE00:
```

```
            if (c==0) state = STATE000;
            else state = STATENULL;
            break;
        case STATE000:
            if (c==1) state = STATE0001;
            else if (c == 0) STATE000;
            else state = STATENULL;
            break;
        }
    }
    if (state == STATE0001)
    {
        // read the nal header
        fread(nal_header, 1, 16, file);
//        nal_header = (fgetc(file)<<24) | (fgetc(file)<<16) |
(fgetc(file)<<8) | fgetc(file);
        // go back to the beginning of the NALU (start code included)
        fseek(file,-4-16,SEEK_CUR);
    }
    else
    {
        memset(nal_header,0,16);
    }
}

void set_nal_extension(suit_nal_extension_t* ext, unsigned char *
nal_header)
{
    unsigned char a, b, c, z;
    if (ext == NULL)
        return;
    a = nal_header[1]; // to be tested
    b = nal_header[2]; // to be tested
    c = nal_header[3]; // to be tested

    z = nal_header[0]; // to be tested

    ext->i_simple_priority_id = a & 0x3F; // 6 bits

    ext->i_temporal_level = (b>>5) & 0x07;  // 3 bits
    ext->i_dependency_id = (b>>2) & 0x07;   // 3 bits
    ext->i_quality_level = (b & 0x03);       // 2 bits

    ext->i_layer_base_flag = (c>>6) & 0x01;             // 1 bit
    ext->i_use_base_prediction_flag = (c>>5) & 0x01;    // 1 bit
    ext->i_discardable_flag = (c>>4) & 0X01;            // 1 bit
    ext->i_fragmented_flag = (c>>3) & 0x01;             // 1 bit
    ext->i_last_fragmented_flag = (c>>2) & 0x01;        // 1 bit
    ext->i_fragment_order = (c & 0x03);                 // 2 bit
}
/*
 Function that try to find the SPS, PPS, and SEI at the beginning of
 the file (until the first coded slice is met).
  */

int getHeaders(FILE* fstream, SuitScalableStream* pstream)
{
    int nalu_cnt=0;
    bool stop_tag = false;
```

```
    resetScalableStream(pstream);
    while(!feof(fstream))
    {
        suit_nal_extension_t extension;
        unsigned char nal_header_current[16];
        unsigned char nal_header_next[16];
        int nal_type;
        int next_nal_type;
        long start,end;
        unsigned char* streamBuffer;
        findNextStartCode(fstream, nal_header_current);
        start = ftell(fstream);

        fseek(fstream,1,SEEK_CUR);
        findNextStartCode(fstream, nal_header_next);
        end = ftell(fstream);
        if (end == start)
        {
            return -1; // failure;
        }
        fseek(fstream,start,SEEK_SET);

        nal_type = (nal_header_current[0] & 0x1F);

        set_nal_extension(&extension, nal_header_current);

        if (nal_type == 6) // SEI of base layer, or SEI wrapping header
of scalable layers
        {
            // does not respect exactly the SEI standard: it should
iterate over the 255 (cf SEI)
            // but actually, no SEI type > 255 exists for the moment (max
around 30)
            if ( nal_header_current[1] == 22) // scalabilty_info SEI,
specific to SVC
            {
                int layer_id;
                int size;
                int i;
                for(i=2;i<16;i++)
                {
                    if (nal_header_current[i] != 0xFF)
                        break;
                }

                layer_id =
nal_header_current[i+1];//extension.i_dependency_id;
                size = pstream->layers[layer_id]->base->size;

                streamBuffer = pstream->layers[layer_id]->base->buffer +
size;
                pstream->layers[layer_id]->base->size += end - start;
                fread(streamBuffer,1,end-start,fstream);
                memset(streamBuffer,0,5+i);
                streamBuffer[5+i] = 1;
            }
            else
            {
                int size = pstream->layers[0]->base->size;
```

```
                streamBuffer = pstream->layers[0]->base->buffer + size;
                pstream->layers[0]->base->size += end - start;
                fread(streamBuffer,1,end-start,fstream);
            }
        }
        else if (nal_type == 20 && extension.i_quality_level == 0) //
spatial NAL
        {
            int layer_id = extension.i_dependency_id;
            int size = pstream->layers[layer_id]->base->size;

            streamBuffer = pstream->layers[layer_id]->base->buffer +
size;
            pstream->layers[layer_id]->base->size += end - start;
            fread(streamBuffer,1,end-start,fstream);
            memset(streamBuffer,0,7);
            streamBuffer[7] = 1;
        }
        else if (   nal_type == 1 // base coded picture AU
                 || nal_type == 2
                 || nal_type == 3
                 || nal_type == 4
                 || nal_type == 5
                 || nal_type == 7
                 || nal_type == 8)// base layer headers
        {
            int size = pstream->layers[0]->base->size;

            streamBuffer = pstream->layers[0]->base->buffer + size;
            pstream->layers[0]->base->size += end - start;
            fread(streamBuffer,1,end-start,fstream);
        }
        else
        {
            // do nothing
            fseek(fstream,end-start,SEEK_CUR);
        }

        nalu_cnt ++;
        // if the next nal unit is not scalable, it is the base layer
        // all the scalable nalu (for a given timestamp) have been
processed
        next_nal_type = (nal_header_next[0] & 0x1F);

        // we stop if the next NAL is a coded slice
        if (   next_nal_type == 1 // coded slice non-IDR
            || next_nal_type == 2 // coded slice data partition A
            || next_nal_type == 3 // coded slice data partition B
            || next_nal_type == 4 // coded slice data partition C
            || next_nal_type == 5)// coded slice IDR)
        {
            if (stop_tag == true)
                break;
            stop_tag = true;
        }

    }
    return 0; // success
```

```
}

/*
*get a new frame for each layer (spatial or snr)
*/
int getNewFrame(FILE* fstream, SuitScalableStream* pstream)
{
    int nalu_cnt=0;
      int bytes_read = 0;
    int au = 0;//access unit

    while(!feof(fstream))
    {
        unsigned char nal_header_current[16];
        unsigned char nal_header_next[16];
        int nal_type;
        long start,end;
        unsigned char* streamBuffer;
        findNextStartCode(fstream, nal_header_current);
        start = ftell(fstream);

        fseek(fstream,1,SEEK_CUR);
        findNextStartCode(fstream, nal_header_next);
        end = ftell(fstream);
        if (end == start)
        {
            return bytes_read; // failure;
        }
        fseek(fstream,start,SEEK_SET);
        bytes_read += end - start;
        nal_type = (nal_header_current[0] & 0x1F);

        if (nal_type == 20) // SNR or spatial scalable NALU
        {
            suit_nal_extension_t extension;
            set_nal_extension(&extension, nal_header_current);

            if (extension.i_quality_level == 0)
            {

                int layer_id = extension.i_dependency_id;
                streamBuffer = pstream->layers[layer_id]->base->buffer;
                pstream->layers[layer_id]->base->size = end - start;
                fread(streamBuffer,1,end-start,fstream);
                memset(streamBuffer,0,7);
                streamBuffer[7] = 1;
            }
            else
            {
                int snr_id = extension.i_quality_level - 1;
                int layer_id = extension.i_dependency_id;
                streamBuffer = pstream->layers[layer_id]->snr[snr_id]-
>buffer;
                pstream->layers[layer_id]->snr[snr_id]->size = end -
start;
                fread(streamBuffer,1,end-start,fstream);
            }
        }
        else if (nal_type == 6)
```

```
        {
            if (nal_header_current[1] == 25)
            {
                // do nothing, this is a quality SEI (specific to SUIT)
                fseek(fstream,end-start,SEEK_CUR);
            }
            else if (nal_header_current[1] == 22)
            {
                // do nothing, this is a upper spatial layer SEI
(specific to SUIT)
                // this SEI contains a SEI, a SPS or a PPS that has
already been decoded
                // while creating the SUIT decoder.
                fseek(fstream,end-start,SEEK_CUR);
            }
            else
            {
                streamBuffer = pstream->layers[0]->base->buffer;
                pstream->layers[0]->base->size += end - start;
                fread(streamBuffer,1,end-start,fstream);
            }
        }
        else// base layer NALU, or sequence headers
        {
            streamBuffer = pstream->layers[0]->base->buffer;
            pstream->layers[0]->base->size += end - start;
            fread(streamBuffer,1,end-start,fstream);
            if(nal_type == 1 || nal_type == 5)//One slice
                au = 1;
        }

        nalu_cnt ++;
        // if the next nal unit is a coded base slice (without
scalability), it is the base layer
        // all the scalable nalu (for a given timestamp) have been
processed
        if ( au &&  ((nal_header_next[0] & 0x1F) == 1
            || (nal_header_next[0] & 0x1F) == 2
            || (nal_header_next[0] & 0x1F) == 3
            || (nal_header_next[0] & 0x1F) == 4
            || (nal_header_next[0] & 0x1F) == 5
            || (nal_header_next[0] & 0x1F) == 7
            || (nal_header_next[0] & 0x1F) == 8  ))
        {
            break;
        }

    }
    return bytes_read; // success
}
```

A.5 Código-fonte da visualização

```
typedef struct SuitRenderer
{
    SDL_Surface *screen;
    SDL_Overlay *overlay;
    FILE* fyuv;
    int width;
    int height;
} SuitRenderer;

void RDR_create(HSuitRenderer* handle, int width, int height, FILE* fyuv)
{
    SuitRenderer* renderer;

    if (handle == NULL || width == 0 || height == 0)
        return;

    if (*handle != NULL) // already allocated
    {
        renderer = *((SuitRenderer**)handle);
    }
    else
    {
        renderer = (SuitRenderer*) malloc(sizeof(SuitRenderer));
        memset(renderer,0,sizeof(SuitRenderer));
        // create SDL window
        if ( SDL_Init(SDL_INIT_VIDEO) < 0 ) {
            printf("Cannot initialize SDL : %s\n", SDL_GetError());
        }

    }

    // output to screan _OR_ to file
    if (fyuv == NULL)
    {
        renderer->screen = SDL_SetVideoMode(width, height, 32,
SDL_SWSURFACE);
        renderer->overlay = SDL_CreateYUVOverlay(width, height,
SDL_YV12_OVERLAY, renderer->screen);
    }
    else
    {
        renderer->fyuv = fyuv;
    }
    renderer->width = width;
    renderer->height = height;

    *handle = renderer;
}

void RDR_delete(HSuitRenderer handle)
{
    SuitRenderer* renderer = (SuitRenderer*) handle;

    if (renderer == NULL)
        return;
```

```
    if (renderer->overlay != NULL)
        SDL_FreeYUVOverlay(renderer->overlay);

    SDL_Quit(); // this function free "renderer->screen" (see SDL
documentation)

    free(renderer);
}


void RDR_blit(HSuitRenderer handle, YuvImage* yuvimage)
{
    int i, j;
    SuitRenderer* renderer;
    SDL_Overlay* overlay;
    FILE* fyuv;
    int width;
    int height;
    SDL_Rect dstrect;

    if (handle == NULL || yuvimage==NULL)
        return;

    renderer = (SuitRenderer*) handle;
    overlay = renderer->overlay;
    fyuv = renderer->fyuv;
    width = renderer->width;
    height = renderer->height;

    dstrect.x = 0;
    dstrect.y = 0;
    dstrect.w = width;
    dstrect.h = height;

    if (overlay != NULL)
    {
        SDL_LockYUVOverlay(overlay);

        for(i=0;i<3;i++)
        {
            // this is due to the inversion between U and V in ffmpeg and
SDL
            int dstI = ((i&1)<<1) + ((i&3)>>1);

            int planeWidth = yuvimage->plane[i].width;
            int planeStride = yuvimage->plane[i].stride;
            int planeHeight = yuvimage->plane[i].height;
            unsigned char* planeAddr = yuvimage->plane[i].addr;
            unsigned char* dst = (unsigned char*)overlay->pixels[dstI];
            int strideDst = overlay->pitches[dstI];
            for(j=0; j<planeHeight; j++)
            {
                memcpy(dst, planeAddr, planeWidth);
                dst += strideDst;
                planeAddr += planeStride;
            }
        }
```

```
        SDL_UnlockYUVOverlay(overlay);

        SDL_DisplayYUVOverlay(overlay, &dstrect);
    }
    if (fyuv != NULL)
    {
        if (   yuvimage->plane[0].width != 0
            && yuvimage->plane[1].width != 0
            && yuvimage->plane[2].width != 0)
        {
            for(i=0;i<3;i++)
            {
                int dstWidth = renderer->width/(i==0?1:2);
                int planeStride = yuvimage->plane[i].stride;
                int planeHeight = yuvimage->plane[i].height;
                unsigned char* planeAddr = yuvimage->plane[i].addr;
                for(j=0; j<planeHeight; j++)
                {
                    fwrite(planeAddr, 1,dstWidth, fyuv);
                    planeAddr += planeStride;
                }
            }
        }
    }
}
```